

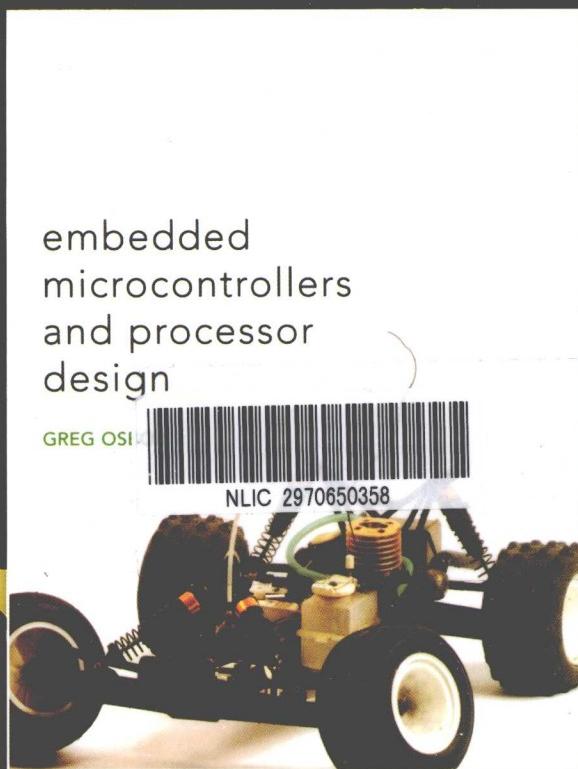
PEARSON

计 算 机 科 学 从 书

# 嵌入式微控制器 与处理器设计

(美) Greg Osborn 著 宋廷强 高树静 译

Embedded Microcontrollers  
and Processor Design



机械工业出版社  
China Machine Press

计 算 机 科 学 从 书

# 嵌入式微控制器 与处理器设计

(美) Greg Osborn 著 宋廷强 高树静 译

Embedded Microcontrollers  
and Processor Design



机械工业出版社  
China Machine Press

本书全面讲述了嵌入式微处理器与微控制器的基础知识，书中没有简单罗列各种嵌入式微控制器的电气特性、物理特性以及具体使用等内容，也没有罗列具体开发工具及开发软件的具体使用，而是关注于让读者理解微控制器背后的基本概念和设计方法，从全局上把握嵌入式微处理器与微控制器的发展、现状以及主要技术等内容。全书内容覆盖嵌入式微控制器、软/硬件调试、模/数转换、外设接口、数字信号处理以及模糊逻辑等主要概念，以便使读者更好地理解和把握嵌入式系统的设计方法和设计理念。

本书强调嵌入式微处理器及微控制器的架构和技术特点，使其更适合用作高校电子电气工程、计算机以及工程技术类相关专业的教材，还可用作专业嵌入式微控制器设计人员的参考书。

Simplified Chinese edition copyright © 2011 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *Embedded Microcontrollers and Processor Design* (ISBN 978-0-13-113041-8) by Greg Osborn, Copyright © 2010.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Prentice Hall.

本书封面贴有 Pearson Education (培生教育出版集团) 激光防伪标签，无标签者不得销售。

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问 北京市展达律师事务所

**本书版权登记号：图字：01-2009-4558**

**图书在版编目(CIP)数据**

嵌入式微控制器与处理器设计/(美)奥斯本(Osborn, G.)著；宋廷强,高树静译. —北京：  
机械工业出版社, 2011.3

(计算机科学丛书)

书名原文：Embedded Microcontrollers & Processor Design

ISBN 978-7-111-32281-8

I . 嵌… II . ①奥… ②宋… ③高… III . ①微控制器 - 系统设计 ②微处理器 - 系统  
设计 IV . TP332

中国版本图书馆 CIP 数据核字 (2010) 第 204132 号

机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码 100037)

责任编辑：张少波

北京市荣盛彩色印刷有限公司印刷

2011 年 4 月第 1 版第 1 次印刷

185mm × 260mm · 22.75 印张

标准书号：ISBN 978-7-111-32281-8

定价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88378991; 88361066

购书热线：(010) 68326294; 88379649; 68995259

投稿热线：(010) 88379604

读者信箱：hzjsj@hzbook.com

文艺复兴以降,源远流长的科学精神和逐步形成的学术规范,使西方国家在自然科学的各个领域取得了垄断性的优势;也正是这样的传统,使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中,美国的产业界与教育界越来越紧密地结合,计算机学科中的许多泰山北斗同时身处科研和教学的最前线,由此而产生的经典科学著作,不仅擘划了研究的范畴,还揭示了学术的源变,既遵循学术规范,又自有学者个性,其价值并不会因年月的流逝而减退。

近年,在全球信息化大潮的推动下,我国的计算机产业发展迅猛,对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇,也是挑战;而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下,美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此,引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用,也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始,我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力,我们与 Pearson, McGraw - Hill, Elsevier, MIT, John Wiley & Sons, Cengage 等世界著名出版公司建立了良好的合作关系,从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum, Bjarne Stroustrup, Brian W. Kernighan, Dennis Ritchie, Jim Gray, Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Abraham Silberschatz, William Stallings, Donald E. Knuth, John L. Hennessy, Larry L. Peterson 等大师名家的一批经典作品,以“计算机科学丛书”为总称出版,供读者学习、研究及珍藏。大理石纹理的封面,也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助,国内的专家不仅提供了中肯的选题指导,还不辞劳苦地担任了翻译和审校的工作;而原书的作者也相当关注其作品在中国的传播,有的还专程为其书的中译本作序。迄今,“计算机科学丛书”已经出版了近两百个品种,这些书籍在读者中树立了良好的口碑,并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑,这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化,教育界对国外计算机教材的需求和应用都将步入一个新的阶段,我们的目标是尽善尽美,而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正,我们的联系方法如下:

华章网站:www.hzbook.com

电子邮件:hzjsj@hzbook.com

联系电话:(010)88379604

联系地址:北京市西城区百万庄南街1号

邮政编码:100037



华章教育

华章科技图书出版中心

嵌入式系统被定义为以应用为中心、以计算机技术为基础,软件和硬件可裁剪,适应应用系统对功能、可靠性、成本、体积及功耗等有严格要求的专用计算机系统。嵌入式系统的应用十分广泛,在消费电子、工业制造、过程控制、网络通信、仪器仪表、汽车、船舶、航空航天及军事装备等方面,都可以找到嵌入式系统的应用。

嵌入式系统的核心是嵌入式微处理器和微控制器,可应用于嵌入式系统的处理器主要有嵌入式微处理器、微控制器、嵌入式 DSP 以及系统集成芯片等。嵌入式微处理器具有体积小、重量轻、成本低、可靠性高的优点,目前常用的 386EX、Power PC、68000、MIPS 和 ARM 等系列都属于嵌入式处理器。嵌入式微控制器又称为单片机或单芯片微控制器,其最大特点是单片化,体积大大减小,片上外设资源丰富。微控制器是目前嵌入式系统工业的主流,占整个嵌入式市场份额的 70%,如 8051、MCS-96、MC68HC05、68300、PIC 等都属于该类型。SoC 技术是目前集成电路制造工艺、计算机技术发展的产物,可以在一个硅片上实现一个更为复杂的系统;各种通用处理器或微控制器都可以以 IP 核的形式作为 SoC 设计的库单元。目前,嵌入式微处理器与微控制器的品种总量已达 1000 多种,有 30 多个流行体系结构,越来越多的公司有自己的处理器设计部门。

本书全面讲述了嵌入式微处理器与微控制器的基础知识,书中没有简单地罗列各种嵌入式微控制器的电气特性、物理特性以及具体使用等内容,也没有罗列具体开发工具及开发软件的具体使用,而是注重于让读者理解微控制器背后的基本概念和设计方法,从全局上把握嵌入式微处理器与微控制器的发展、现状以及主要技术等内容。全书内容覆盖嵌入式微控制器、软/硬件调试、模数转换、外设接口、数字信号处理以及模糊逻辑等主要概念,以便使读者更好地理解和把握嵌入式系统的设计方法和设计理念。本书强调了嵌入式微处理器及微控制器的架构和技术特点,适合用作高校电子电气工程、计算机以及工程技术类相关专业的教材,还可用作专业嵌入式微控制器设计人员的参考书。本书的最大特色如下。

- 覆盖面广。本书内容主要与基于微控制器的设计相关,内容围绕微控制体系结构、单片机和嵌入式 IP 核展开,不仅涵盖了主要芯片以及 IP 核,还重点介绍了微控制体系结构的相关知识。
- 内容新颖。本书将传统的嵌入式处理器与当前一些新兴技术有机结合,包括 SoC 设计、IP 核、在线调试、嵌入式 DSP、模糊控制等内容,充分归纳一些技术的共性,力求呈现给读者浅显易懂的设计理论与方法。
- 实用性强。书中很多内容都基于实例进行讲解,其中参考的单片机都是典型的商用经典设计,微控制器核都是已经发行的 SoC IP 核以及市场上的相关产品。每一章都在开始给出章节学习目标,除第 1 章外,其余各章在最后都给出习题,便于读者检查学习效果。

本书由我和高树静博士共同翻译完成,由我负责全书的统稿工作。对于本书的出版我们首先要感谢机械工业出版社的编辑,是他们的努力促成了本书的顺利翻译与发行,使读者能够通过本书及时了解嵌入式微控制器相关技术;同时也感谢李鹏程、江依妹、姚永、张朝阳、菊勇、刘洪涛、林家希等给予本书的核对与检查。

在本书的翻译过程中,我们力求忠实于原著,但由于译者技术和翻译水平有限,对一些词句把握不够准确,书中难免存在错误和疏漏之处,敬请读者批评指正,以便在后期修改完善。我们的邮箱是 songtq@163.com,敬请赐教。

宋廷强

2010 年 10 月于青岛

如今,微控制器已经成为人们日常生活中普遍应用的设备,我们日常使用的大多数电气产品中都内置有微控制器,在家用电器、汽车、复印机、移动电话,甚至强大的机车控制中,都可以找到它们的身影。只要有电气使用的地方,就可以发现微控制器的使用!

以微控制器为主题的书籍有很多,为什么还要再写一本新书呢?所有流行的微控制器芯片和体系结构都有关于“怎样使用”的书籍出版,而本书关注的是向读者全面介绍微控制器技术,既包括单片机,也包括作为知识产权(Intellectual Property, IP)核形式的微控制器。

很多电子工程专业的学生需要使用微控制器来学习嵌入式系统设计方面的课程,Intel 8051、ZiLOG Z8 或者 Microchip PIC 等器件十分流行,也是同学们学习微控制器的首要选择。另外,这些器件还对一些使用广泛并且十分廉价的开发工具提供设计支持。

基于嵌入式微控制器的设计选择十分广泛,这是工程师们今天所要面对的问题。他们不仅要选择流行的单片机,还要选择用于 ASIC 片上系统(System-on-a-Chip, SoC)设计所要用到的 IP 核。尽管在计算机领域基于 Intel 的体系结构处于统治地位,微控制器领域仍然靠创新设计来发展。

本书围绕 3 个主要内容展开讲解——微控制器体系结构介绍、单片微控制器和嵌入式 IP 核。每一章都在开始给出章节学习目标,并在各章节最后给出习题(除第 1 章外),用以检查学习效果。

本书不仅涵盖了主要芯片以及 IP 核,还重点介绍了微控制器体系结构的概念。例如,计算器件是怎样演变的以及在设计中为什么要使用不同类型的器件。

本书参考的单片机都是典型的商用经典设计。当然,很多其他设计也可以利用,尤其是来自无晶圆设计公司的设计。微控制器核参考了已经发行的 SoC IP 核以及市场上的相关产品。本书关于体系结构的介绍中,“处理器”这一概念既包含“单片微控制器”的“处理器”元素,也包含“IP 核”的概念。

本书希望能向读者提供单片机或嵌入式微控制器及微处理器设计的相关知识。书中讨论了 CISC 和 RISC 处理器之间的差异,也介绍了单片机设计流程和嵌入式微处理器设计流程。

本书对 16 位 Freescale MC9S12X 系列单片微控制器进行了详细介绍,同样,还详细介绍了基于 RISC 结构的 PIC18F4520 和 ZiLOG Z8 微控制器。书中还介绍了 8 位微控制器,并详述了许多控制器系列的大量外设。

书中介绍了指令集体系结构(Instruction Set Architecture, ISA)的概念,以便于更好地理解基于 CISC 和 RISC 体系结构处理器的共性,并扩展到基于使用 ARM 和 MIPS 指令集体系结构的 IP 核的嵌入式 SoC 微控制器设计。书中详细介绍了 ARM10TDMI 和 MIPS32 4KE<sup>TM</sup> IP 核。

可配置处理器技术越来越重要,尤其是在设计高性能消费类电子产品时更是这样。可配置处理器技术允许定制微处理器核,而该微处理器核的配置会对 SoC 嵌入式设计的性能和功耗带来影响。书中介绍了 Tensilica Xtensa LX2 系列可配置处理器。

书中讨论了由 RISC 派生的专用处理器。对数字信号处理器(Digital Signal Processor, DSP)进行了概述,包括 Texas Instruments 公司的 TMS320C55 处理器和 Analog Devices 的 ADSP-BF533 Blackfin 处理器。书中介绍了相关工程设计流程的方法,讨论了可供工程师设计开发的不同工具,给出了使用集成设计环境(Integrated Design Environment, IDE)开发单片机的实例。

微控制器设计的软件编程既可以像草坪洒水车控制编程一样简单,也可以像控制机器人的

RTOS 一样复杂。从简单的轮询到复杂的多级中断系统,这些编程技术在书中都有论述。许多单片机都有串行 I/O 接口功能模块,它们主要用于数据通信。书中介绍了 UART、I<sup>2</sup>C、I<sup>2</sup>S、CAN/LIN SPI 以及 USB 等外设功能模块。

SoC 设计需要与半导体制造厂家紧密结合。作为无晶圆设计技术,SoC 需要专门的工程技术将需要的功能模块集成到芯片中。IP 功能模块既可以由集成电路设计厂家提供,也可以从那些独立的设计公司获得,但是得到一款能够正常工作的芯片仍是一个复杂的过程。

本书倾向于对单片机和嵌入式形式微控制器的介绍和理解。ISA 的概念与产品设计方法一起讲解,通过 IP 核的使用来引入 SoC 设计的概念。

从任何抽象层次上来说,微控制器设计都是基于现有可用技术的折中。本书关注的 3 个基本技术是处理器、存储器和软件:处理器技术根据半导体制造厂家的能力而定,存储器技术采用层次化存储结构实现,软件技术实现依赖于汇编和优化编译技术。

就本书内容所覆盖的范围来说,概述与基于微控制器的设计特性相关。一般地,基于 CISC 的微处理器比基于 RISC 的微处理器的指令更为复杂,RISC 的寄存器组与 CISC 相比是正交的,RISC 的 C 语言优化编译器比 CISC 的效率更高。

RISC 和 CISC 是全球范围广泛使用的指令集体系结构。指令集体系结构的一些创新,如 VLIW 和 EPIC 等,本书通过比较的形式进行了介绍。书中重点关注了微控制器技术是基于 RISC 的,还是基于 CISC 的,这样可以为读者理解其他演化出来的指令集体系结构提供必要的基础知识。

微控制器的核心是微处理器。本书中,处理器具有广义的含义。不管是 SoC 设计中以 IP 核形式实现,还是传统的单片机形式,其基本的处理器概念是相同的。MIPS32 4KE™ IP 核可以制作成 NEC 单片机,也可以制作成 CISCO SoC 路由器;其实现不同,但都具有相同的体系结构。

本书旨在能够为在校工程专业学生介绍微控制器技术相关概念,而不是作为硬件参考手册,也不是作为一系列的应用指南。书中给出的概念具有一般形式,这样可以使更大范围工程专业的学生理解基本的概念,并将其应用到实际中。

作者十分感谢下述人员对于本书书稿进行的阅读和校验:杨百翰大学的 C. Richard G. Helps、伊利诺伊学院的 James Streib、德福瑞大学哥伦布分校的 Chao-Ying Wang 和佛蒙特州技术学院的 Richard Warren。

Greg Osborn

# 目 录

Embedded Microcontrollers and Processor Design

出版者的话

译者序

前言

第 1 章 嵌入式处理器	1
1.0 微控制器	1
1.1 微控制器市场	1
1.2 数据路径	1
1.3 商用微控制器	2
1.4 SoC 内核处理器	2
1.5 SoC 单元相对销售量	2
1.6 超大规模集成电路(VLSI)芯片 设计工具	3
1.7 IP 核	4
1.8 指令集体系结构	4
1.9 投资与回报	4
1.10 半导体技术的发展	5
参考文献	7
第 2 章 微控制器体系结构	8
2.0 单片计算机	8
2.1 约翰·冯·诺依曼	8
2.2 计算机体系结构	9
2.3 半导体技术	9
2.3.1 小规模集成电路	10
2.3.2 硬件总线	10
2.3.3 智能外围接口	10
2.3.4 标准 I/O 接口	10
2.4 MSI 和 LSI	11
2.5 电子计算器	12
2.6 微处理器	12
2.6.1 应用型数据处理	13
2.6.2 Intel i4004	13
2.6.3 Intel i8080	13
2.7 微处理器外设	14
2.8 Intel i8051 微控制器	15
2.9 RISC 简介	16

2.9.1 RISC 处理器	16
2.9.2 RISC 的协同作用	16
2.9.3 RISC 市场	17
2.10 无晶圆半导体公司	17
2.10.1 RISC IP 核	17
2.10.2 RISC 工艺流程	18
2.11 嵌入式控制器 IP 核	18
2.11.1 CISC IP 核	19
2.11.2 RISC IP 核	19
2.11.3 第三方 IP 核	19
2.12 专用处理器	19
2.13 本章小结	20
习题	20
参考文献	21
第 3 章 嵌入式微控制器技术	22
3.0 集成电路	22
3.1 摩尔定律	22
3.1.1 微处理器的性能	22
3.1.2 实现技术	23
3.1.3 阿姆达尔定律	24
3.1.4 技术融合	24
3.2 设计抽象	25
3.2.1 指令集体系结构	25
3.2.2 处理器家族	26
3.3 RISC 和 CISC	26
3.3.1 处理器技术	26
3.3.2 性能评估	26
3.3.3 程序指令	27
3.3.4 指令成本	27
3.3.5 微代码指令	27
3.4 存储器技术	28
3.4.1 局部性	28
3.4.2 存储器分级	29
3.4.3 高速缓存	29
3.4.4 一级缓存和二级缓存	30
3.4.5 数据寄存器	30

3.4.6 指令队列	30	4.1.1 CMOS 晶体管	44
3.4.7 分支指令	30	4.1.2 CMOS 功耗	45
3.4.8 存储器访问延迟	31	4.1.3 封装	45
3.4.9 高速缓存模块	31	4.1.4 工作温度范围	46
3.5 指令处理	32	4.2 存储器工艺	46
3.5.1 汇编语言	33	4.2.1 DRAM	46
3.5.2 程序编译器	33	4.2.2 SRAM	46
3.5.3 硬编码指令	34	4.2.3 NVRWM	47
3.6 程序设计	34	4.2.4 EEPROM	47
3.6.1 程序代码大小变化	34	4.2.5 Flash 工艺	48
3.6.2 CISC 指令集	34	4.2.6 ROM	48
3.7 统一指令集	35	4.3 硬件特性	48
3.7.1 工业标准软件	35	4.3.1 配置字	48
3.7.2 指令集扩展	35	4.3.2 振荡器类型	49
3.8 RISC 指令集体系结构	35	4.3.3 复位	49
3.8.1 微代码	36	4.3.4 待机模式	50
3.8.2 微指令周期	36	4.3.5 低功耗	50
3.8.3 专用指令	36	4.3.6 看门狗定时器	50
3.8.4 单周期指令	36	4.3.7 在线编程	51
3.9 处理器逻辑	37	4.4 数据输入/输出	51
3.9.1 同步逻辑	37	4.4.1 并行 I/O	51
3.9.2 寄存器堆	38	4.4.2 三态 I/O 引脚	52
3.9.3 正交寄存器	38	4.4.3 内存映射 I/O	52
3.9.4 寄存器优化	38	4.5 同步串行通信	52
3.9.5 载入/存储数据操作	38	习题	53
3.10 处理器功能划分	38	参考文献	53
3.10.1 指令流水线	39	第 5 章 程序设计	54
3.10.2 执行单元	39	5.0 程序设计	54
3.10.3 流水线级	39	5.1 轮询程序	54
3.10.4 流水线吞吐量	40	5.1.1 程序流程	54
3.10.5 顺序执行	40	5.1.2 程序时序	55
3.10.6 分支执行	40	5.1.3 连续任务	55
3.11 五级流水线	41	5.1.4 任务时序	56
3.11.1 指令流水线阻塞	42	5.1.5 连续多任务	56
3.11.2 分支预测表	42	5.2 中断	57
3.11.3 数据流水线阻塞	42	5.2.1 异步时序	57
3.12 本章小结	42	5.2.2 中断允许	57
习题	43	5.2.3 机器状态	58
参考文献	43	5.2.4 延时	58
第 4 章 微控制器功能	44	5.2.5 上下文切换	58
4.0 设备功能	44	5.2.6 中断向量	59
4.1 晶体管工艺	44	5.2.7 中断嵌套	59

5.2.8 关键代码 .....	60
5.2.9 中断服务程序 .....	61
5.3 实时操作系统 .....	61
5.4 事件驱动系统 .....	61
5.5 内核 .....	62
5.6 系统分层 .....	62
5.7 风险 .....	62
习题 .....	63
参考文献 .....	63
第6章 软/硬件调试 .....	64
6.0 软/硬件调试 .....	64
6.1 COTS 控制器工具 .....	64
6.2 嵌入式控制器工具 .....	65
6.3 首款芯片 .....	66
6.4 板级探针 .....	66
6.5 调试步骤 .....	67
6.5.1 软件编辑 .....	67
6.5.2 编译 .....	68
6.5.3 程序生成 .....	68
6.5.4 仿真器 .....	69
6.5.5 在线仿真 .....	70
6.6 SoC 调试策略 .....	70
6.6.1 SoC 软件调试 .....	70
6.6.2 内核调试 .....	71
6.6.3 JTAG/EJTAG 规范 .....	72
6.7 ARM SoC 调试 .....	72
6.8 MIPS SoC 调试 .....	73
习题 .....	74
参考文献 .....	75
第7章 串行数据通信 .....	76
7.0 串行数据通信 .....	76
7.1 UART .....	77
7.1.1 异步模式 .....	77
7.1.2 发送/接收缓冲器 .....	78
7.2 串行外围接口 SPI .....	79
7.3 I <sup>2</sup> C 总线 .....	81
7.3.1 I <sup>2</sup> C 总线如何工作 .....	82
7.3.2 I <sup>2</sup> C 总线术语 .....	84
7.3.3 总线传输术语 .....	85
7.4 CAN 总线 .....	85
7.5 LIN 网络 .....	88
7.6 I <sup>2</sup> S 总线 .....	88
7.6.1 I <sup>2</sup> S 串行数据 .....	89
7.6.2 I <sup>2</sup> S 字选择 .....	89
7.6.3 I <sup>2</sup> S 总线时序 .....	90
7.7 IrDA .....	90
7.8 USB 总线 .....	91
7.8.1 USB 拓扑 .....	91
7.8.2 USB 构架 .....	92
7.8.3 USB 物理连接 .....	93
7.8.4 USB 接口 .....	93
7.8.5 USB 2.0 规范 .....	93
7.9 蓝牙 .....	95
7.9.1 蓝牙构架 .....	95
7.9.2 蓝牙频率 .....	95
7.9.3 蓝牙网络 .....	96
习题 .....	96
参考文献 .....	97
第8章 模数转换 .....	98
8.0 模数转换 .....	98
8.1 模数转换概述 .....	98
8.2 换能器 .....	99
8.3 低通滤波器 .....	100
8.4 采样 .....	101
8.5 香农采样定理 .....	102
8.6 什么是模数转换器 .....	102
8.6.1 ADC 的分辨率 .....	103
8.6.2 LSB 和 MSB 定义 .....	103
8.6.3 量化 .....	103
8.6.4 量化误差 .....	104
8.6.5 偏置误差 .....	106
8.6.6 微分非线性 .....	106
8.6.7 丢码 .....	107
8.6.8 信噪比 .....	107
8.7 模数转化算法 .....	108
8.7.1 逐次逼近 .....	109
8.7.2 SAR ADC 结构 .....	110
8.7.3 Flash ADC .....	111
8.7.4 集成 ADC .....	111
8.7.5 流水线 ADC .....	113
8.7.6 Σ - Δ 转换器 .....	114
8.8 过采样 .....	114
习题 .....	115
参考文献 .....	116

第9章 数字信号处理 .....	117
9.0 数字信号处理 .....	117
9.1 什么是 DSP .....	117
9.1.1 滤波与合成 .....	118
9.1.2 DSP 性能 .....	119
9.1.3 模拟信号转换 .....	119
9.2 DSP 控制器构架 .....	119
9.3 模拟滤波器 .....	121
9.3.1 滤波性能测试 .....	122
9.3.2 时域响应 .....	123
9.3.3 模拟低通滤波器 .....	123
9.3.4 有源模拟滤波器 .....	124
9.3.5 有源滤波器的比较 .....	124
9.4 数字滤波器 .....	125
9.4.1 FIR 滤波器 .....	125
9.4.2 FIR 滤波器的实现 .....	126
9.4.3 卷积 .....	127
9.4.4 IIR 滤波器 .....	128
9.5 信号变换 .....	129
9.5.1 相量模型 .....	129
9.5.2 傅里叶级数 .....	130
9.5.3 离散傅里叶级数 .....	130
9.5.4 傅里叶变换 .....	130
9.5.5 离散傅里叶变换 .....	130
9.6 快速傅里叶变换 .....	132
9.6.1 FFT 的执行 .....	132
9.6.2 DFT 蝶形变换 .....	133
9.7 表寻址 .....	133
习题 .....	134
参考文献 .....	134
第10章 模糊逻辑 .....	136
10.0 模糊逻辑 .....	136
10.1 模糊逻辑方法 .....	137
10.2 模糊感知 .....	138
10.3 模糊逻辑的术语 .....	138
10.4 模糊专家系统 .....	139
10.4.1 推理过程 .....	140
10.4.2 模糊化 .....	140
10.4.3 推理 .....	141
10.4.4 合成 .....	141
10.4.5 去模糊化 .....	142
10.5 语言变量 .....	142
10.5.1 使用语言变量 .....	143
10.5.2 模糊规则剖析 .....	144
10.5.3 语言变量的逻辑组合 .....	144
10.6 PID 控制器 .....	144
10.6.1 时间语言变量 .....	145
10.6.2 语言变量比较 .....	145
10.7 模糊逻辑应用 .....	146
10.8 规则矩阵 .....	147
10.8.1 模糊逻辑的实现 .....	147
10.8.2 隶属函数 .....	149
10.8.3 隶属度输入 .....	151
10.8.4 推理 .....	152
10.9 去模糊化 .....	152
10.10 调整与提升系统性能 .....	153
习题 .....	153
参考文献 .....	154
第11章 8位微控制器 .....	155
11.0 通用微控制器 .....	155
11.1 微芯公司 PIC18F4520 .....	156
11.1.1 PIC18F4520 Harvard 体系结构 .....	157
11.1.2 指令流水线 .....	157
11.1.3 特性 .....	158
11.1.4 电源管理模式 .....	158
11.1.5 振荡器配置 .....	159
11.1.6 复位 .....	159
11.1.7 存储器组织 .....	160
11.1.8 中断结构 .....	161
11.1.9 输入/输出(I/O)端口 .....	162
11.1.10 定时器相关的功能 .....	164
11.1.11 定时器模块 .....	164
11.1.12 采样/比较/PWM 功能 .....	167
11.1.13 串行通信接口 .....	169
11.1.14 模数转换 .....	173
11.1.15 模拟比较器 .....	174
11.1.16 CPU 特性 .....	174
11.1.17 指令集 .....	175
11.1.18 电特性 .....	176
11.2 ZiLOG Z8 ENCORE! XP F0830 系列 .....	176
11.2.1 eZ8 CPU 描述 .....	177
11.2.2 Z8 Encore! CPU 体系结构 .....	178

11.2.3 地址空间 ······	179	12.2.1 低功耗设计 ······	234
11.2.4 外设概述 ······	180	12.2.2 灵活的时钟系统 ······	235
11.2.5 复位控制器和停止模式恢复 ······	182	12.2.3 MSP430 CPU ······	235
11.2.6 低功耗模式 ······	182	12.2.4 操作模式 ······	236
11.2.7 通用输入/输出 ······	183	12.2.5 FLL <sup>+</sup> 时钟模块 ······	237
11.2.8 中断控制器 ······	184	12.2.6 Flash 存储控制器 ······	238
11.2.9 定时器 ······	185	12.2.7 硬件多路器 ······	239
11.2.10 Watchdog 定时器 ······	189	12.2.8 DMA 控制器 ······	240
11.2.11 模数转换器 ······	189	12.2.9 数字 I/O ······	240
11.2.12 比较器 ······	190	12.2.10 Watchdog 定时器 ······	240
11.2.13 Flash 存储器 ······	191	12.2.11 定时器 A 和 B ······	241
11.2.14 非易失性数据存储 ······	192	12.2.12 USART ······	242
11.2.15 片上调试器 ······	192	12.2.13 USCI ······	243
11.2.16 振荡器控制 ······	193	12.2.14 ADC12 的功能 ······	246
11.2.17 eZ8 CPU 指令和编程 ······	194	12.2.15 DAC12 模块 ······	248
习题 ······	196	12.2.16 嵌入式仿真模块 ······	248
参考文献 ······	196	习题 ······	250
<b>第 12 章 16 位微控制器 ······</b>	<b>197</b>	参考文献 ······	250
12.0 16 位处理器概述 ······	197	<b>第 13 章 知识产权 SoC 核 ······</b>	<b>251</b>
12.1 Freescale S12XD 处理器概述 ······	197	13.0 SoC 概述 ······	251
12.1.1 XGATE 概述 ······	199	13.1 SoC 设计挑战 ······	252
12.1.2 时钟 ······	203	13.1.1 可配置处理器 ······	253
12.1.3 模/数转换器(ATD) ······	205	13.1.2 SoC 综合 ······	254
12.1.4 增强型捕捉定时器(ECT) ······	206	13.1.3 可扩展处理器 ······	255
12.1.5 脉宽调制(PWM) ······	207	13.1.4 可扩展处理器替代 RTL ······	256
12.1.6 I <sup>2</sup> C 总线 ······	209	13.1.5 清晰的控制方案 ······	257
12.1.7 CAN 总线 ······	209	13.2 MIPS32 4K 处理器核系列 ······	258
12.1.8 串行通信接口(SCI) ······	211	13.2.1 4KE 系列的主要特点 ······	259
12.1.9 串行外围接口(SPI) ······	215	13.2.2 执行单元 ······	261
12.1.10 定时中断定时器(PIT) ······	217	13.2.3 乘除单元(MDU) ······	262
12.1.11 电压调整器(VREG) ······	217	13.2.4 内存管理单元(MMU) ······	263
12.1.12 背景调试模块(BDM) ······	218	13.2.5 cache 控制器 ······	264
12.1.13 中断模块(XINT) ······	219	13.2.6 总线接口单元(BIU) ······	264
12.1.14 映射存储器控制(MMC) ······	221	13.2.7 电源管理 ······	265
12.1.15 调试(DBG) ······	222	13.2.8 指令 cache ······	265
12.1.16 外部总线接口 ······	224	13.2.9 数据 cache ······	266
12.1.17 端口综合模块 ······	224	13.2.10 EJTAG 控制器 ······	266
12.1.18 2K 字节 EEPROM(EETX2K) ······	227	13.2.11 系统协处理器 ······	267
12.1.19 512K 字节 Flash 模块 (FTX512K4) ······	229	13.2.12 用户自定义指令(UDI) ······	267
12.1.20 安全性 ······	230	13.2.13 指令流水线 ······	267
12.2 Texas Instruments MSP430 <sup>TM</sup> 系列 ······	231	13.2.14 指令 cache 失效 ······	269
12.2.1 低功耗设计 ······	234	13.2.15 数据 cache 失效 ······	269

13.2.16	乘法/除法操作	270	14.1.11	使用自动生产的处理器快速 进行 SoC 开发	297
13.2.17	分支延迟	270	14.1.12	起点:基本的接口和计算	298
13.2.18	内存管理	271	14.1.13	并行处理任务	298
13.2.19	操作模式	271	14.1.14	自动指令集发生的含义	301
13.3	ARM1022E 处理器概述	273	14.2	Tensilica Xtensa 体系结构概述	301
13.3.1	处理器组成	275	14.3	指令集设计原则	303
13.3.2	寄存器	275	14.4	Tensilica Xtensa 处理器的 独有特性	303
13.3.3	整数核	275	14.5	寄存器	304
13.3.4	整数核流水线	277	14.6	指令长度	305
13.3.5	内存管理单元	279	14.7	复合指令	306
13.3.6	cache 和写缓冲	280	14.8	分支	306
13.3.7	总线接口	280	14.9	指令流水线	308
13.3.8	拓扑结构	281	14.10	有限的指令常数宽度	308
13.3.9	协处理器接口	281	14.11	短指令格式	309
13.3.10	协处理器流水线	282	14.12	寄存器窗口	309
13.3.11	调试单元	282	14.13	Xtensa L2 总结	310
13.3.12	挂起模式	283	习题		310
13.3.13	监视器调试模式	283	参考文献		310
13.3.14	时钟和 PLL	283	第 15 章 数字信号处理器		311
13.3.15	ETM 接口逻辑	283	15.0 DSP 概述		311
13.3.16	工作状态	284	15.1 TMS320C55x		311
13.3.17	状态转换	284	15.1.1 TMS320C55x 的特性		312
13.3.18	在异常处理中切换状态	285	15.1.2 C55x 的主要特征		313
13.3.19	工作模式	285	15.1.3 指令集体体系结构		313
习题		15.1.4 主要功能单元		315	
参考文献		15.1.5 特殊属性		322	
第 14 章 Tensilica 可配置 IP 核		15.1.6 低功耗设计		322	
14.0 简介:再谈摩尔定律		15.1.7 处理器片上外设		323	
14.1 芯片设计工艺		15.1.8 仿真和测试		330	
14.1.1 设计错误的芯片		15.2 Analog Devices 公司 ADSP-BF535 Blackfin 处理器		331	
14.1.2 SoC 设计的基本趋势		15.2.1 便携低功耗体系结构		331	
14.1.3 每个系统都采用一个新的 SoC 实现是不现实的		15.2.2 系统集成		331	
14.1.4 纳米技术		15.2.3 处理器核		332	
14.1.5 SoC 设计改革		15.2.4 存储器体系结构		335	
14.1.6 SoC 可编程性		15.2.5 事件处理		337	
14.1.7 可编程性与效率对比		15.2.6 DMA 控制器		339	
14.1.8 SoC 设计成功的关键		15.2.7 外部存储控制		340	
14.1.9 改进的设计方法学用于 SoC 设计		15.2.8 异步控制器		341	
14.1.10 可配置处理器作为构建 模块		15.2.9 PCI 接口		341	

15.2.10	USB 设备	342
15.2.11	实时时钟	342
15.2.12	Watchdog 定时器	343
15.2.13	定时器	343
15.2.14	串口	344
15.2.15	串行外设接口(SPI)端口	345
15.2.16	UART 端口	346
15.2.17	动态电源管理	346
15.2.18	工作模式和状态	347
	习题	347
	参考文献	348

# 嵌入式处理器

- 本章目标：微控制器相关概念简介

- 主要内容：

1. 微控制器的基本应用及其市场情况
2. 商用嵌入式处理器的概念
3. 采用 IP 核的 SoC 系统芯片设计
4. 指令集体体系结构的概念
5. 半导体技术发展趋势

## 1.0 微控制器

微控制器主要应用在三个方面：作为独立设备；作为嵌入式处理器系统的一个部分；集成到 SoC 中。在每种层次的应用中，微控制器都将会通过逻辑模块来整合附加的特征，以增强可用性。而对于 SoC 来说，系统中的所有逻辑单元将被整合到一个单独的集成电路中（见图 1-1）。无论是通用芯片还是独立芯片，通常都会采用最广泛的专用特征集，这样才能提高潜在的市场吸引力。

## 1.1 微控制器市场

大多数商用微控制器是面向多种应用而设计的，这样就能够拓宽产品的市场，同时也能降低成本。图 1-2 中列出了微控制器应用的十二个主要市场，每一个条目下又包含了多种不同的应用。

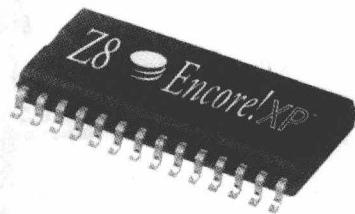


图 1-1 ZiLOG eZ8 处理器  
(经 ZiLOG 公司许可使用)

航空及国防电子	数字成像
汽车	工业测量与控制
电池供电	医疗电子
广播与娱乐	服务器输入/输出
消费/互联网应用	电信
数据通信	无线

图 1-2 微控制器主要市场

## 1.2 数据路径

说到微控制器，通常要说到其数据路径的位宽。数据路径位宽指的是从存储器并行取出的数据位数或算术逻辑单元（Arithmetic Logic Unit, ALU）并行处理的数据位数。微控制器的数据路径位宽和嵌入式处理器内核位宽通常是 4 位、8 位、16 位、32 位或 64 位。更宽的字长，像 128 位或者 256 位，一般出现在专用处理器（Application Specific Processor, ASP）中，比如用来

支持视频游戏的图像控制器中所采用的处理器。

### 1.3 商用微控制器

从数量上看，绝大多数微控制器是基于8位数据路径的。一些商用微控制器采用了8位微处理器内核和一些基本的功能模块。个别早期的芯片设计则成为了事实上的工业标准，Intel i8051和Freescale 68HC11就是典型例子。

图1-3反映了采用微芯科技公司的PIC微控制器进行设计的可能应用范围。从图中可以看出，仅仅是这些外围接口就可以用于设计一些简单或复杂的嵌入式系统控制器。SoC设计则通过将CPU和外设整合到一个晶片上从而将设计提高到了另一个层次。

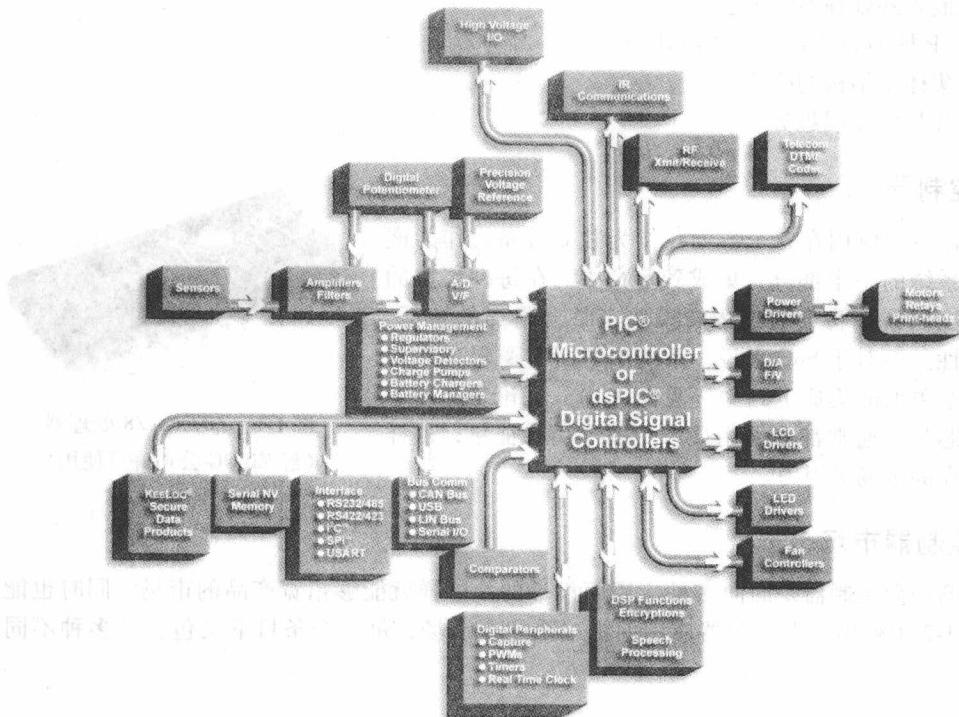


图1-3 微芯片 PIC 嵌入式控制解决方案（经微芯科技股份有限公司许可使用）

### 1.4 SoC 内核处理器

图1-4表示了SoC设计中嵌入式处理器内核的销售量，这些内核的销售量是极高的，反映出其市场极为广阔。采用4位微控制器的廉价电子腕表销售量已达千万件，而稍贵一些的32位视频游戏控制台的生产也包含数以百万的内核单元。从全球来看，4位或8位处理器的销售总量每年都超过10亿件。

### 1.5 SoC 单元相对销售量

商用嵌入式处理器的总体市场分布如图1-5所示。图中将市场划分为5个主要部分，在饼图中所占的百分比相当于其市场份额，从图中我们不难看出每个应用领域的市场份额。微控制器可以跨领域应用，这就表明其设计能够面向十分广阔的市场。

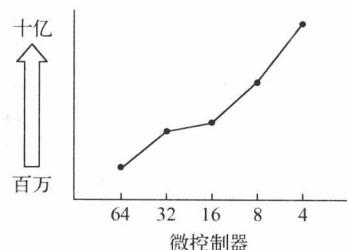


图1-4 嵌入式处理器销售量

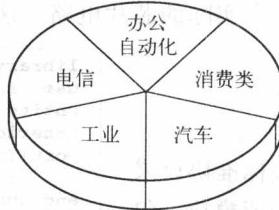


图 1-5 市场细分的美元份额

## 1.6 超大规模集成电路 (VLSI) 芯片设计工具

随着半导体制造工艺的进步，微芯片可以集成数以百万的晶体管。目前，在计算机辅助设计（Computer-aided Design, CAD）工具（见图 1-6）的帮助下，设计工程师已经能够针对特定应用量身打造微芯片。这类微芯片被称为专用集成电路（Application Specific Integrated Circuit, ASIC）。

先进的 VLSI CAD 工具能够将普通微控制器内核合并到芯片设计本身中。这种内核采用了适当的功能模块，专为特定应用而设计。而这些功能模块一旦与处理器内核相连接，就能模拟出计算机系统的基本功能（见图 1-7）。这样的设备就称为 SoC 计算机。

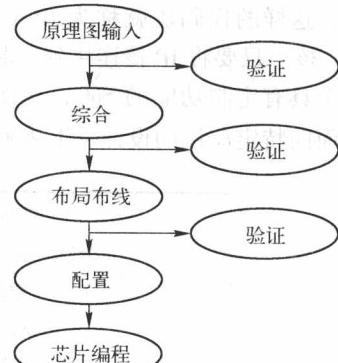


图 1-6 VLSI 设计流程

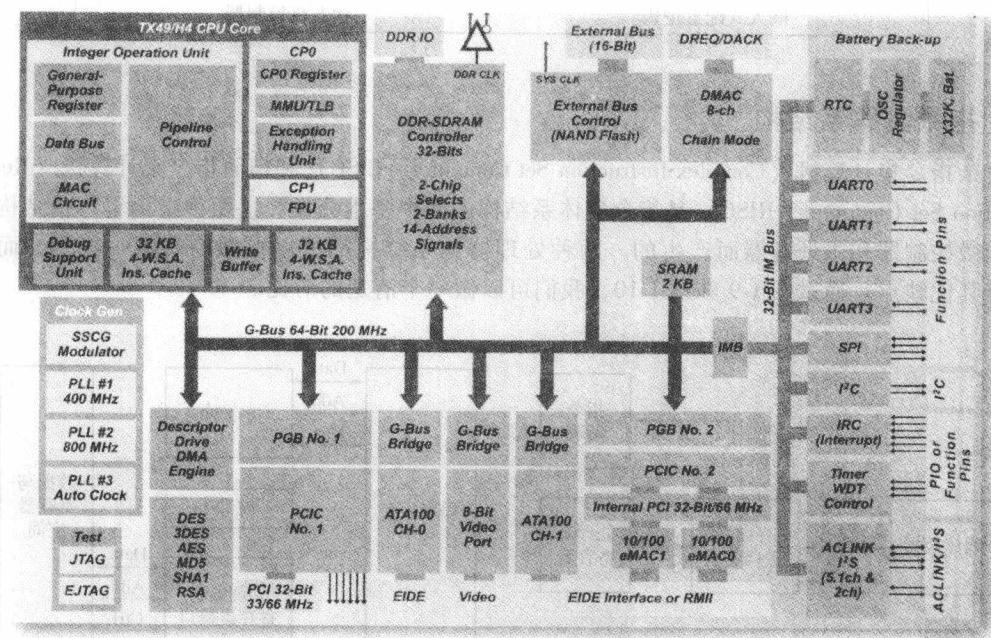


图 1-7 Toshiba TX4939XBG SoC（经东芝美国电子公司许可使用）

设计一个 ASIC 或 SoC 芯片须在一些比较成熟的自动化设计软件的帮助下才能完成。电路可以使用高级编程语言（High-level Programming Language, VHDL）<sup>①</sup>来描述（见图 1-8），而后，计

<sup>①</sup> VHDL 通常是指 Very-High-Speed Integrated Circuit Hardware Description Language，即超高速集成电路硬件描述语言，用于描述硬件电路。——译者注

计算机就会执行这些程序，之后生成一个实际的芯片电路。这是一个复杂的过程，但是从本质上说，微芯片却是来自于软件程序的。

## 1.7 IP 核

软件工程师们已经制定出了一些标准协议来定义晶体管电路，这样，基本逻辑功能模块，包括控制器和核心处理器就可以用独立的代码块来定义，这样的代码块就称为 IP (Intellectual Property) 核。只要将 IP 核连接到一起，就能够设计出一个具有定制功能的 SoC，从而可以以低成本完成面向特定应用的设计。下表列出了常用的 IP 核类型及其应用范围。

```
library IEEE;
use IEEE.std_logic_1234.all;
entity and1 is
generic (gate_delay : Time := 500 ps);
port (In1, In2 : in std_logic;
      Z : out std_logic;
end and1;
architecture behavioral of and1 is
begin
Z <= (In1 and In2) after gate_delay;
end behavioral;
```

图 1-8 通用 VHDL 代码

总线接口	数字信号处理
模拟/混合信号	处理器与微控制器
算术及数学函数	存储器元件
测试内核	安全与误差检测
视频/图像/音频	有线通信
无线通信	平台级 IP
硬件操作系统/实时操作系统	片内通信
嵌入式配置逻辑	状态机控制器

## 1.8 指令集体系结构

复杂指令集计算机（Complex Instruction Set Computer, CISC）和精简指令集计算机（Reduced Instruction Set Computer, RISC）是指令集体系结构的两个基本分类。这两种指令集体系结构都是因设计微控制器和微处理器而产生的。两种处理器体系结构都可以在微控制器中找到，而每一种各有其优缺点。通过图 1-9 和图 1-10，我们可以做一个清楚的对比。

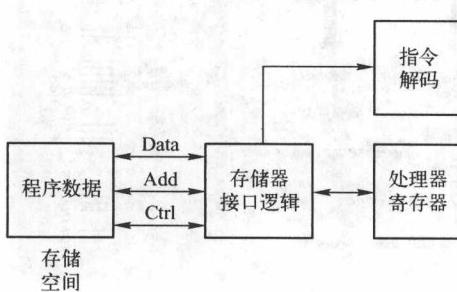


图 1-9 CISC 体系结构

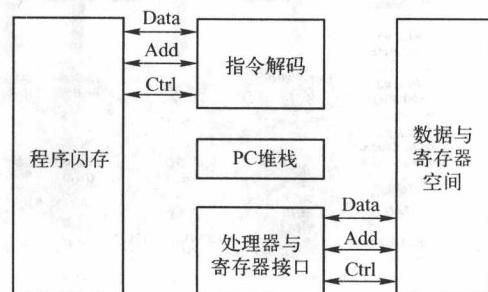


图 1-10 RISC 体系结构

## 1.9 投资与回报

是否采用商用微控制器进行产品开发，需要综合考虑相关的技术因素和经济因素，而经济因素基本上是最重要的因素。商用微控制器大大降低了前端设计的成本，使得产品能够更快进

入市场，一旦产品得以销售，将会给企业带来丰厚的利益。

相比之下，传统的基于 SoC 的设计需要大量的前期投资，产品进入市场需要 18 个月或者更长时间。虽然生产时的成本低于商用微控制器，但是由于需要很长的开发时间，对企业盈利会有很大的影响。

图 1-11 中的曲线图反映了与基于微控制器的产品设计相关的经济指标。这是一个投资回报 (Return on Investment, ROI) 的简单关系图，却有效地揭示了“风险”的概念。图中，横轴下方表示成本，上方表示收益。

采用市场上已有的商用微控制器进行设计可以使产品更快地进入市场。但这样一来，终端产品的成本也会相应提高。因而，这样的设计会挤压企业的利润空间。

采用基于 SoC 的解决方案，前期设计成本会更高，既包括基本的设计成本，也包括在上市时间上所产生的无形风险。基于 SoC 的设计通常要花费一年或更多时间才能上市，而在此期间其他富有竞争力的产品可能正推向市场。因而，正确地认识风险是得到高回报的基础。

很多情况下，是从市场上获得单独部件还是自行设计一个 SoC，在技术上并没有什么太大的区别，这时候就须市场部门来做决定。一般地，能够最快上市并且成本最低的方案才会受到青睐。

## 1.10 半导体技术的发展

半导体技术的发展，或者更确切地说，CMOS 技术的发展速度一直以来都维持在一个较快的水平。半导体工艺技术上的进步，使得 CMOS 晶体管的尺寸不断减小。图 1-12，给出了从头发到电子半径的大致尺寸，以帮助读者有一个清晰的对比。

戈登·摩尔得出一个定律，即 CMOS 晶体管的密度每 18 个月翻一番。这条定律对于半导体及其他相关技术同样适用。对于微处理器（及嵌入式控制器），此定律可以表述为单个芯片上集成的 CMOS 晶体管数量每 18 个月翻一番。摩尔定律如图 1-13 所示。

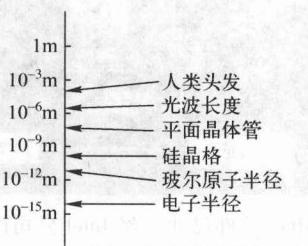


图 1-12 物体的相对尺寸

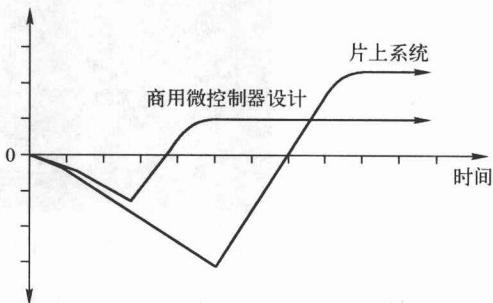


图 1-11 ROI 商用微控制器

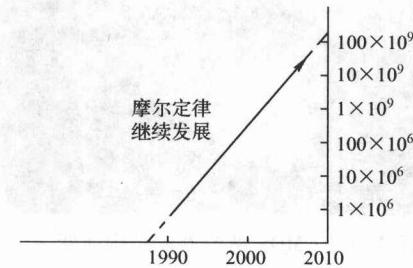


图 1-13 摩尔定律

图 1-14 给出了 CMOS 晶体管尺寸缩小的情况。预计到 2010 年，晶体管门的尺寸将达到 15 纳米级，然而芯片密度将会每 18 个月翻一番，这就意味着，一个芯片上将有数以十亿计的晶体管。当然，这并非简单地在掩膜板上绘制一些更细的线条，许多技术都需要进一步细化才能完成一个 15 纳米级的 CMOS 门。

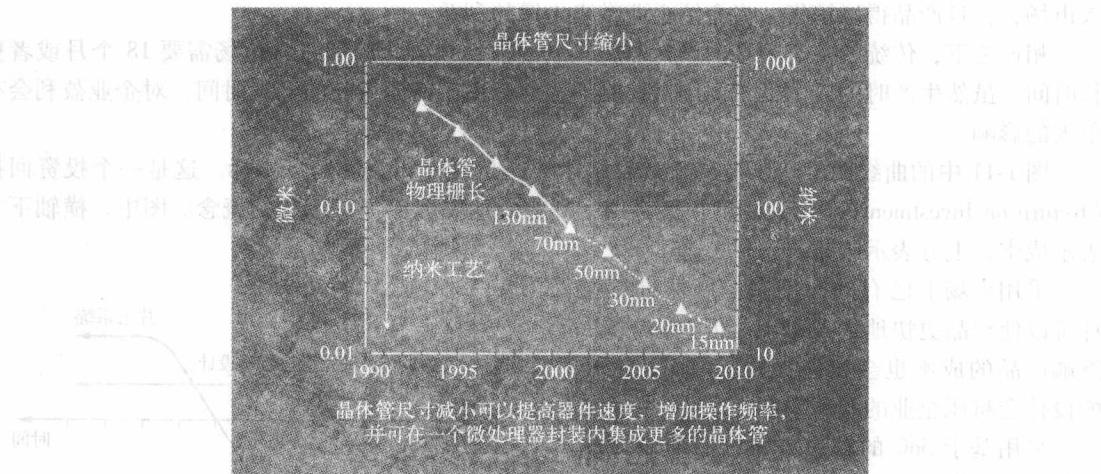


图 1-14 晶体管尺寸缩小示意（经 Intel 公司许可使用）

图 1-15 所示为一个基本的 CMOS 晶体管，栅极、源极和漏极已给出标注。图 1-16 所示为某种类型晶体管的结构演变过程，为了达到 2020 年的技术目标，该类型晶体管的发展是极有必要的。我们可以看到，图中是一个循环上升的设计结构，这表明，CMOS 晶体管的基本几何尺寸也需要随着不断降低的纳米级一同改变。

图 1-17 所示为 Intel 公司提供的一份图表，它表明到 2013 年 CMOS 器件的尺寸需要新技术的支持才能满足摩尔定律的要求。这些技术包括纳米级工艺的生产设备和设计技术，在这一层面上的技术进步需要多种相关技术的共同进步。

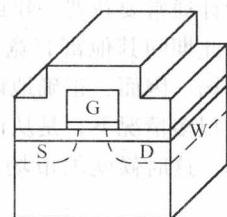


图 1-15 CMOS 晶体管

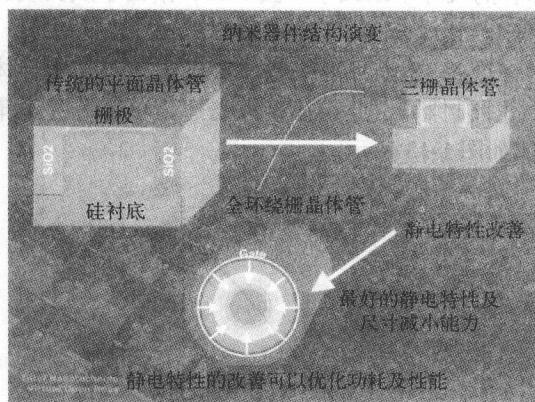
图 1-16 CMOS 纳米器件结构演变  
(经 Intel 公司许可使用)

图 1-17 CMOS 器件尺寸（经 Intel 公司许可使用）

图 1-18 表明了 CMOS 设计技术的未来发展方向。器件大小将不足以限制高性能 CMOS 晶体管的生产，而新的集成解决方案将是不可或缺的。从图 1-19 中我们可以发现，CMOS 晶体管可能最终会达到一个极限，即它们的尺寸不会再小了，Intel 预测将在 2020 年达到该极限。那时，基于量子效应的器件可能就该登场了。从 1971 年引入 i4004 微处理器开始，Intel 作为 CMOS 工艺的先驱，仍然有一段很长的路要走。

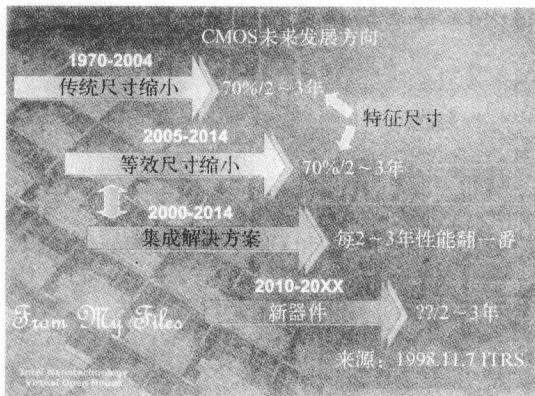


图 1-18 CMOS 蓝图 (经 Intel 公司许可使用)

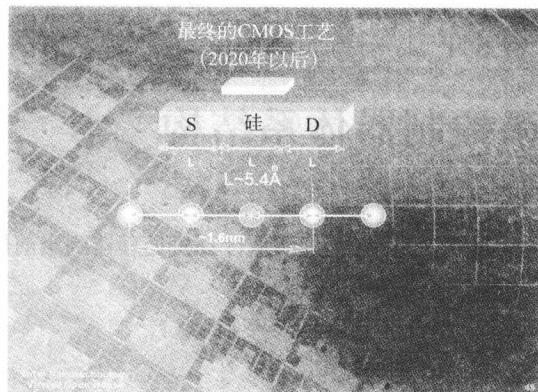


图 1-19 最终的 CMOS (经 Intel 公司许可使用)

## 参考文献

- Gargini, P. *Intel nanotechnology overview presentation*. International Nanotechnology Virtual Open House, Intel Corporation.
- David, K. October 2004. *Silicon nanotechnology presentation*. Intel Corporation.

## 第2章 |

### 嵌入式 Microcontrollers and Processor Design

# 微控制器体系结构

- 本章目标：理解从计算机处理器到当前用于嵌入式产品的 RISC 和 CISC 处理器的演化过程
- 主要内容：
  1. 半导体技术的基本概念
  2. 微处理器概念
  3. 微控制器功能设计
  4. RISC 微控制器介绍
  5. RISC 和 CISC 处理器 IP 核

## 2.0 单片计算机

当前的微控制器和嵌入式处理器的体系结构直接继承了最早的计算机功能。随着时间的推移以及芯片设计和半导体制造技术的发展，这些体系结构已形成了当前的实现方式。由于包含了从早期计算机所继承来的功能，可以真正地称它们为单片计算机。

最早的商业微控制器出现在 20 世纪 80 年代中期，来自诸如 TI、Intel、Fairchild 和 National Semiconductor 等公司。它们由当时已经出现的控制器直接集成，采用标准晶体管 - 晶体管逻辑 (Transistor-Transistor Logic, TTL) 门级电路设计。人们需要采用一种更加灵活的、可编程的、基于逻辑的设计来代替硬连线的固定设计，这是推动其发展的主要动力。而这种发展可以使控制器的成本更低，上市时间更快，也意味着可以获得更高的收益率。

微控制器的设计变化反映出电子产品功能需求的不断变化。随着新的产品不断推向市场，对控制逻辑创新的需求也不断面临新的挑战，人们总是期望电子产品的功能不断增强，价格却不断降低。

一款微控制器对于任何给定设计的可适用性，依赖于许多相互关联的因素，而这些因素并非都是技术性的。在选择适用于某一个产品的微控制器时，经济因素一直以来都是需要考虑的，即使不是最重要的，也是关键的一个因素。然而对于工程师来说，还要考虑的任务是，对于给定的功能描述，如何给出最好的设计。

事实上，微控制器和嵌入式处理器可以且也正被用于所有的机电产品，从日常家居用品到移动电话，再到复杂的工业设备。基于电子技术的控制器可以消除或降低机械复杂性，而这种复杂性很容易导致错误的发生。使用全固态设计可以增强控制器的功能，提高其可靠性，同时减少维护和产品费用。

## 2.1 约翰·冯·诺依曼

约翰·冯·诺依曼（见图 2-1）被认为是现代计算机设计的奠基人之一。与其他许多先驱者一样，他具有坚实的数学基础，在 1928 年获得数学博士学位。由于欧洲战争的爆发，与许多同时代的科学家一样，他移民到美国，并在 Princeton 大学的应用数学学院任职。

第二次世界大战需要每一个人的参与，也包括那些从事技术工作的人。一项紧迫的需求摆在了学术界的面前，需要将他们的理论研究



图 2-1 约翰·冯·诺依曼  
(www.lanl.gov)

转化成实际的应用，以支持国家的安全防卫。冯·诺依曼对于专用数学的研究非常有兴趣，热衷于将数学的严谨应用到新兴的计算机上。

### 冯·诺依曼体系结构

许多科学家团体工作于不同的新技术领域。冯·诺依曼知道如何应用最新开发的电子计算机来解决实际问题，其对于如何将新的电子技术应用于实际计算机的真知灼见被称为“冯·诺依曼体系结构”（见图 2-2）。

这种结构是一个基本概念，规定的是一个程序的指令如何被组织成串行的模式，并分别存储。计算机具有分离的功能单元，并同步运转。冯·诺依曼设计了一个指令系统，能够控制固定连线逻辑的计算机。这些指令将决定计算机要执行的操作。

直到那个时候，计算机对于某一组特定数据的操作都是手工连线，或者固定编码。要改变一个指令序列或者程序，意味着重新连接功能单元之间的连线。这是一项繁琐的、易于出错的任务，也非常耗时，并且严重限制了计算机处理工作的效率。

冯·诺依曼的方法能够提高计算机的有效工作时间（正常运行时间），因此大大提高了其工作效率。由于修改指令代码很容易，使得开发和测试更复杂的程序成为可能。这对于电子计算机新的应用领域的开发是一个开创性的巨大进步。

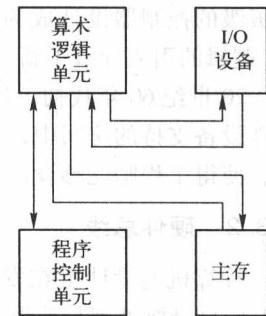


图 2-2 冯·诺依曼体系结构

## 2.2 计算机体系结构

当前的微处理器和微控制器同出一源，随着时间的推移，它们伴随设计中的新技术和创新以及新市场发展至今。然而，它们的一个基本共性是，潜在的变化已经导致数以千计不同芯片设计的出现。

### CISC 和 RISC

CISC 和 RISC 是处理器的两个主要类型。这些指令集体系结构对计算机系统的设计进行了定义。在以前，将一百万个晶体管放在一个芯片中是不可能实现的，由于半导体技术的快速发展，使得新型处理器设计成为可能。

本书将集中讨论具有典型的 CISC 或 RISC 指令集体系结构的嵌入式处理器和微控制器。而实际上，它们具有相同的构造，都执行指令。

在实际应用中，对产品收益率最大化的追求决定了一种实际产品应选择的嵌入式处理器类型。根据这个原则，还需要进行工程设计的折中决策，设计的目标仍然是最强大的功能、最高的性能以及最低的产品成本（见图 2-3）。

上述的两种指令集体系结构，都可以用于单独的商业微控制器，都可用于各种类型产品的知识产权核。对于某个特定的产品设计，没有固定的设计规则来规定选择使用哪种指令集体系结构。在许多的市场领域中，这两种指令集体系结构是相互交叠的，而通常的设计选择仅仅依赖于经济因素。

另外，单片微控制器通常具有更加通用的特性，能够广泛适用于各种市场。基于 IP 的设计倾向于针对范围较窄的某种产品，但它们仍然同时具有单片计算机所必需的基本功能设计元素。

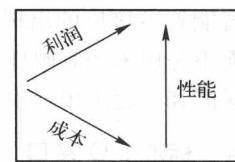


图 2-3 市场影响力

## 2.3 半导体技术

在 20 世纪 60 年代的 10 年中，半导体技术得到了极大的发展，可以将大量的双极型晶体管

设计在一个芯片中，形成逻辑功能单元或者集成电路（Integrated Circuits, IC）。在单个芯片中可以具有 AND、NAND、NOR、FLIP-FLOPS 以及许多其他的逻辑功能，通过将各种逻辑功能单元组合在印制电路板上，可以设计出更大、更复杂的电路。

这种逻辑单元的早期应用主要是设计与计算机相连的设备控制器，计算机的运行速度比与其相连的输入/输出设备的速度要快得多。该电路是用专用电路设计的，受公司产权保护，不能用于商业目的。

### 2.3.1 小规模集成电路

小规模集成电路（Small-Scale Integration, SSI）（见图 2-4）逻辑被推向市场后，使得印制电路板级的控制器设计成为可能。工程师可以采用商业化的组成部件，进行基于晶体管的设计。SSI 逻辑的引入导致了新一代创新的出现。

20 世纪 60 年代初，计算机（主机）的体积是非常大的，也非常笨重，它们需要放置在具有特殊设备支持的房间中，并且房间通常是锁门的，被幽默地称为“寺庙”。商业化逻辑电路的出现，使得工程师能够设计出在尺寸上更加紧凑，并仍然具有庞大主机基本功能的新一代计算机。

### 2.3.2 硬件总线

计算机的设计也在发生着变化。在传统的技术中，中央处理单元（Central Processing Unit, CPU）自身要处理与外设进行接口所需的控制功能。CPU 在同一时刻只能处理一项任务，由于要控制外设，就意味着只能用较少的时间来执行用户程序。因此引入了一种新的设计思想，将系统根据功能进行划分，而不是将所有的控制功能集中在 CPU 上，这样外设的功能就能得到加强，并具有一定的智能性。

总线技术使上述思想成为可能。一个总线就是一个电子“高速公路”，可以在功能单元之间传送信息（数据、地址和控制信号）（见图 2-5）。总线的概念是计算机体系结构中的一项重大创新，现在，所有的微控制器产品都将总线用于它们的设计中。

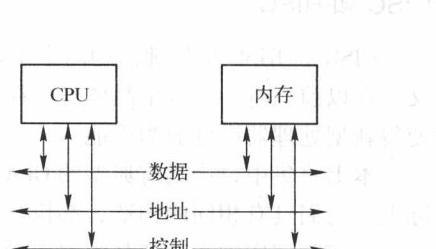
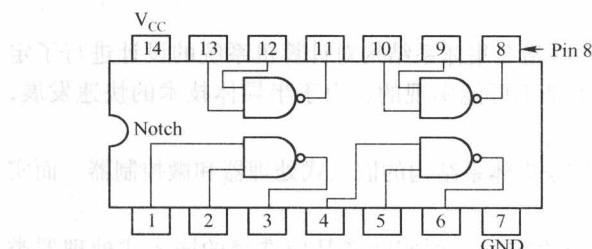


图 2-4 小规模集成电路

图 2-5 基本总线结构

### 2.3.3 智能外围接口

智能外围接口的出现提高了计算机的总体性能。它允许 CPU 将更多的时间用于处理用户程序，而不是用来控制硬件设备。这主要是通过两个时间的比值来体现的，即 CPU 在用户模式的执行时间和在超级用户模式的执行时间，该时间也包括外设的控制时间。

### 2.3.4 标准 I/O 接口

根据市场需求，控制器的设计具有更大的灵活性。如果制造商能够开发出更加灵活、更加通用的控制器，便可以制造尽量少的印制电路板（Printed Circuit Board, PCB），从而降低成本。为了达到这个目的，一种做法是在 PCB 板级创建一个基本核，然后仅使用简单的开关和跳线，根据特定的设备进行设置。PCB 板仍然是一个独立的单元，能够降低生产费用，而且可以为一款特

定的产品进行定制。这种做法类似于当前 PC 主板的实现方式。

直到现在，计算机仍然是一个比较复杂的系统，集成了所有硬件和软件，I/O、内存、磁盘以及其他功能单元都与 CPU 一起紧密地集成在整个硬件设计中。尽管计算机的逻辑功能有基本的功能块设计，但它们仍然是各自独立的集成单元。

第三方电子技术的出现，使得独立设计产品成为可能。这方面最先开始的工作是 I/O 功能，例如终端和打印机，尽管它们具有相应的功能，但是还必须与计算机进行交互。为了实现这种模式，需要有第三方供应商，对特定计算机生产商的接口进行电气和时序特性的匹配。

显然，每一个生产商都具有不同的接口，这是每一个公司所特有的。对一个独立的供应商来说，这就意味着为每一个接口设计一个专用的硬件控制器。尽管存在着一种推动力，使供应商达成一个公用的标准，但是还是需要处理一些公司的专用设计问题（见图 2-6）。

工程师所开发的是智能外围接口。这些智能外设是特殊设计的控制器，能够识别不同的接口，并根据这些接口的差异进行调整。随着计算机系统设计越来越复杂，更多传统 CPU 控制的 I/O 操作被下移到外围设备控制器中。计算机正在使用分块设计技术进行构建，如图 2-7 所示。

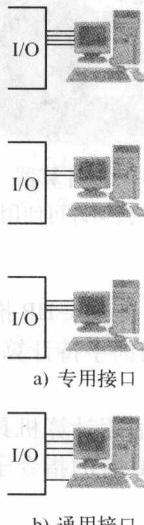


图 2-6

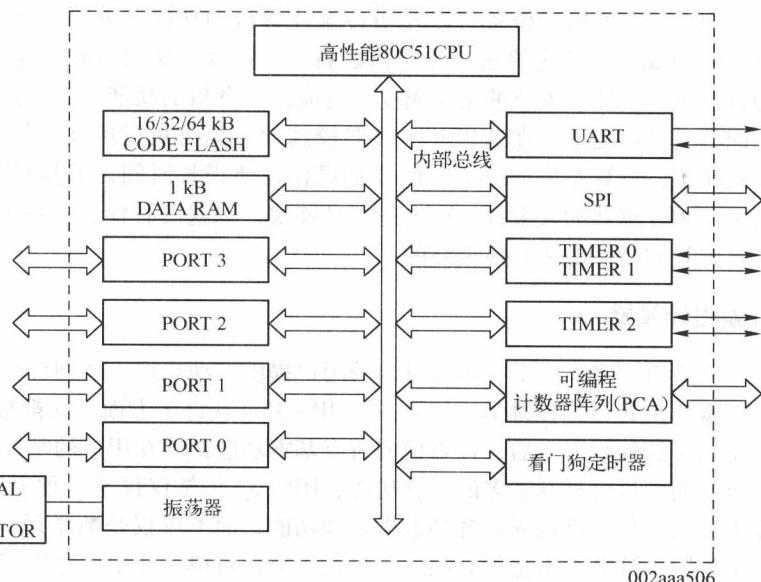


图 2-7 分块设计概念（经 NXP 半导体公司许可使用）

## 2.4 MSI 和 LSI

20世纪60年代末，中规模集成电路（Medium-Scale Integration, MSI）被引入市场。MSI 仍然是基于双极型晶体管逻辑的，在单芯片中，扩展了SSI所能提供的功能。八进制锁存器、4位 ALU 和 16位 NAND 门降低了设计中所需要的 SSI 封装数量。与先前基于 SSI 的控制器相比，可以设计出具有更高复杂性和智能性的产品。

有了这些基础，就可以设计专用的单芯片部件。随后开始出现了标准的接口芯片，例如用于串行通信的 RS-232（见图 2-8）。随着晶体管尺寸的不断缩小，半导体制造工艺技术已经可以直接实现更加复杂的集成电路部件。

根据各种外设功能设计出专用的设备控制器，这种设计的特点是更加复杂，可靠性更高，最重要的是更加经济；所设计的控制器不仅针对传统的键盘、

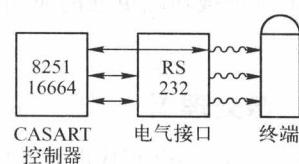


图 2-8 RS-232 串行 I/O 接口

阴极射线管（Cathode Ray Tube, CRT）显示器以及打印机，也针对许多计算机相关的其他功能。同时，新的组成部件也开发了出来，以加强计算机的控制功能，例如直接内存访问、内部时钟、时序电路，以及新开发的串行通信协议等。

还有另外一个方向可以发展。更高的晶体管集成度意味着对于一个特定的设计，只需要更少的组成部件来实现。组成部件的减少，可以降低控制器产品设计部分的费用，发展的趋势就是在单芯片中实现这些控制器。

## 2.5 电子计算器

1968年，TI推出了一种4功能单芯片电子计算器（见图2-9）。它支持加、减、乘、除四种运算，极大地吸引了公众的注意力。自此，单芯片计算器诞生了。该计算器除降低了传统计算器的成本之外，还开启了便携式计算器的新时代。由于单芯片所需要的功耗降低，显示的功耗也降低，所以计算器可以采用电池供电。

现在，小的电子设备可以模仿传统计算机的功能。桌面计算机具有两种或三种形式的输入和输出设备——键盘、显示器和可操作的打印机，直接与人类的反应速度相适应。计算机的功能就在于此，不同的只是数据率。在这里并不需要较高的计算机执行速度。与计算机相比，计算器执行两个数的相加操作需要很长时间，但是只要对于使用计算器的人来说，这个速度足够快，也就没有什么关系了。能够感觉到的性能才是最重要的。

### 可编程计算器

从逻辑上来说，下一步就可以为用户提供自动执行一系列计算的功能。1971年初，HP推出了一款便携式可编程科学计算器——HP-35。在许多方面，它都很像是一个缩微的手持计算机。当添加了软件程序之后，计算机的所有基本功能都尽在用户的掌控之中。

然而，尽管具有诸多的计算功能，HP-35仍然仅仅是一种复杂的计算器。通用计算机具有庞大的指令集，可以支持各种数据处理功能，而不仅仅是数学运算。HP-35所具有的指令子集非常有限，计算能力也很有限。它只是一个针对单一应用的专用产品。

20世纪70年代末和80年代初，半导体技术的发展非常迅速，出现了大规模集成电路（LSI）。新开发的N沟道和P沟道金属-氧化物半导体（Metal-oxide Semiconductor, MOS）的使用，可以将数千个晶体管在一个芯片中实现集成。作为这种技术的一个推动力，就是要替代速度慢的、笨重的且易于出错的计算机内存。在1971年，出现了具有1K位存储量的设备——INTEL1101。

这种新技术使得工程师能够设计出复杂并且功能强大的部件。此外，市场的推动作用也促使工程师们设计出更加先进、更加成熟、更加经济的控制器。一旦部件的性能增强了，自然就可以承载功能更强的应用程序。技术的进步不仅满足了当前的需求，也为新的设计创造了机会。

## 2.6 微处理器

自20世纪80年代以来，计算机越来越向着用户友好型发展，对CRT的需求也呈戏剧性的增长势头。每一生产商都有其特定的需求，计算机I/O系统直接控制终端，而终端也有固定的规定。此外，由于终端的应用越来越广泛，采用中央I/O控制器来直接控制终端也变得越来越困难。

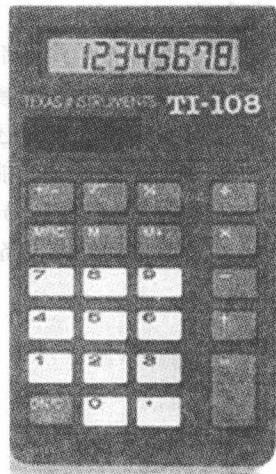


图2-9 4功能计算器  
(经德州仪器公司许可使用)

从 I/O 控制器中移出一部分标准控制功能到终端是有必要的。这样一来，终端控制器就负责实现监测键盘按键、将按键信号发送给计算机、在 CRT 上显示字符等功能，如图 2-10 所示。

ASCII 字符集最初是工业上用来对常用字符进行编码的，包含大小写字母、数字、标点等。采用 7 位二进制数足够表示 128 ( $2^7$ ) 个基本的字符，并且可以方便地包含在一个字节内，如图 2-11 所示。

CRT 生产厂商希望半导体产业能够为他们提供一个更好的解决方案。随着大规模集成电路集成逻辑功能不断增强，工程师们能够开发出更加复杂的控制器。CRT 生产厂商所需要的就是一款可以满足他们生产需要的可编程控制器，正如数学上所用的可编程计算器。

### 2.6.1 应用型数据处理

半导体生产厂商收到了来自产品开发工程师们的需求，核心需求是一个针对其专业应用的可编程器件。事实上，他们都需要一个针对他们产品的计算器。

随着需求的不断增长，越来越多面向应用的大规模集成电路控制器也投入到了市场中。这些需求有一个共同之处，即都需要输入/输出和控制功能，就像计算器芯片一样。HP - 35 可以广泛应用于数学问题中去，因为它拥有此类功能，并具有可编程能力。

半导体生产厂商清楚地知道，在新的大规模集成电路技术的支撑下，他们可以生产出面向各种应用的可编程器件。半导体厂商只需在设计中整合更多的指令，生产出的芯片就可以适用于多种领域，就是说，他们的产品将更加通用化。

但是这也会带来芯片生产成品和售价的提高。然而，应用的灵活性在这里远比成本更重要。相比固定的专用指令集，这种新产品拥有更多通用指令。只要将芯片接到 LSI 存储器和 I/O 设备上，它就能应用到特定的环境中去。

### 2.6.2 Intel i4004

CRT 生产厂商需要一款能够处理 4 位数据的控制器，以便处理 ASCII 字符集。CRT 厂商希望借此处理从键盘扫描到的二进制信号，这些信号不仅需要以字符形式显示出来，还要传递到 CPU 中进行处理。Intel 公司为此专门开发了一种新的部件，即 Intel i4004，其芯片组由三芯片组成；虽然体积很小，却能够处理 4 位数据，是一款地道的微处理器，如图 2-12 所示。

### 2.6.3 Intel i8080

i8080 处理器，顾名思义，它能够处理 8 位数据。这里所指的“数据”是来自电路其他部分的信号，它们被翻译成 0 和 1 并转换成为字节格式。i8080 对通过它的数据进行了一个信息的转换，如图 2-13 所示。

i8080 需要其他一些支持芯片作为补充，比如程序存储器、数据存储和 I/O 端口，当然也包括系统控制逻辑，如图 2-14 所示。使用微处理器也会带来其他方面的影响，比如它增加了印刷版的尺寸和复杂程度，使得设计过程更为复杂。

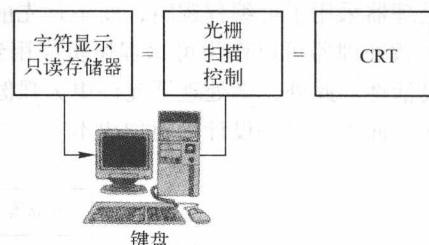


图 2-10 终端设计

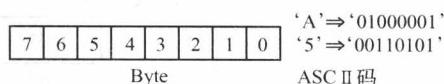


图 2-11 ASCII 字节编码

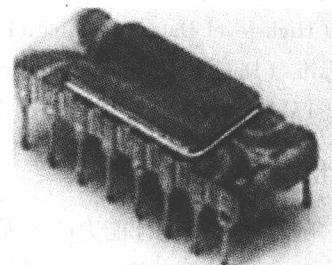


图 2-12 Intel i4004  
(经 Intel 公司许可使用)

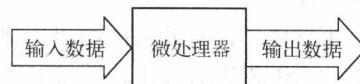


图 2-13 微处理器转换功能

以上因素都会增加控制器的成本，但是它给设计工程师带来的方便足以补偿成本上的增加。微处理器采用了可编程逻辑，而不是先前的固定逻辑。

微处理器可以作为可编程逻辑应用到 CRT 终端中去，从而大大提高终端针对不同接口配置的灵活性。此外，微处理器允许更为强健的特征集，可以为不同客户量身定制，更加关注客户的需求，而不仅仅是设计的附加成本。

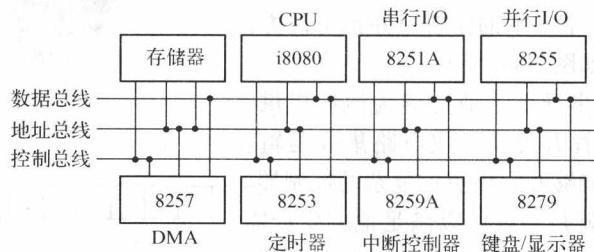


图 2-14 Intel i8080 系统芯片组

微处理器在嵌入式控制器中的最初应用是极为成功的。于是，微处理器的应用很快便拓展到了各种设计领域，以通过其可编程性来提高产品的性能。由于客户可以根据需要拓展产品的功能，产品设计变得更加以市场为导向。

## 2.7 微处理器外设

微处理器的成功带动了市场对相关部件的需求。但是，微处理器的支撑电路也遇到了瓶颈。微处理器取代了固定逻辑设计，与之相适应，市场需要越来越多 LSI 部件，以简化附加的逻辑电路。

进行基于微处理器特性的设计时，有些方面需要保持一致。i8080 基于 8 位的字长，自然地，相应的芯片应该具有 8 位数据端口，比如 i8255，它的每个输入/输出端口都可编程。

对于 CRT 终端来说，串行 I/O 接口是必不可少的；i8251 USART (Universal Synchronous/Asynchronous Receiver/Transmitter) 满足了这种需求。为了满足计数及计时的需要，i8253 也已开发成功。此外，随着工程师对于各种附加功能需求的增加，高层数据连接控制/同步数据连接控制 (High-level Data Link Control/Synchronous Data Link Control, HDLC/SDLC) 协议芯片、数据加密标准 (Data Encryption Standard, DES) 加密芯片及其他高度集成的可编程设备也已问世。图 2-15 列出了部分常用的外围设备，它们都能与 Intel 公司的产品相兼容。

## 微计算机

随着工程师设计能力的增强，从系统级来看，控制器也开始具有了一些计算机的特点。控制器中整合了多种功能，包括程序和数据存储、I/O、计时器和计数器以及系统控制逻辑，如图 2-16 所示。事实上，由于它们是可编程的，所以它们可以通过其他方式得到应用，比如处理数据。从某种程度上来讲，第一台 CRT 终端控制器扮演着计算机的角色，体积很小，自然地可以称它们为微计算机。

微计算机设计已经经受住了市场的考验，而工程师们对微计算机的需求也在不断增长。一些产品，比如便携式设备，对控制器的体积有较大限制。此外，一些设备仅仅需要简单的 I/O 及其他功能。

半导体厂商对此挑战给出了满意的回应。依照他们的观点，基于微处理器的系统可以减小到单一 LSI 芯片的尺寸。目前半导体工艺技术的水平限制了产品所能达到的层次，然而，从功能上来说，基本的单芯片微处理器系统是有可能实现的。

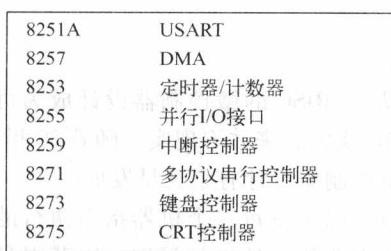


图 2-15 Intel 外围芯片

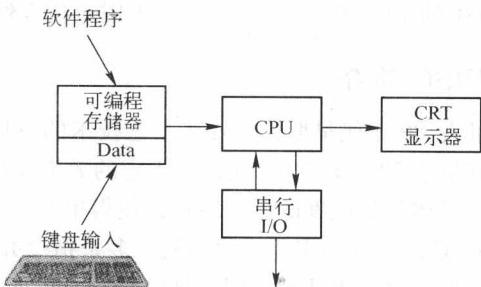


图 2-16 智能控制器

## 2.8 Intel i8051 微控制器

i8080 的外围部件，作为实际的系统构建模块，最终成为了事实上的工业标准。通过减少存储器的容量和简化外设功能，设计一个单芯片微控制器是可行的方案。这样，i8051 便在 1977 年上市了，如图 2-17 所示为早期的 i8051 (MCS-51) 功能模块表。

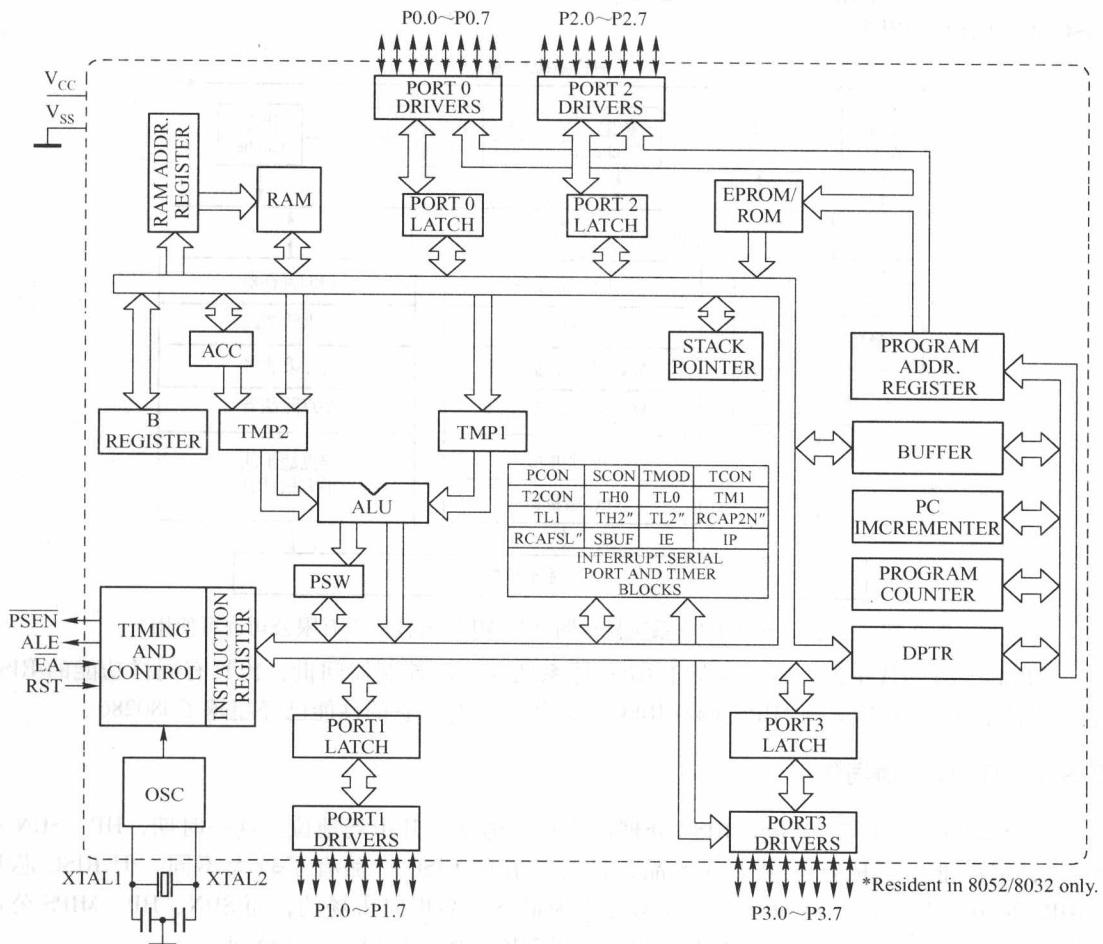


图 2-17 i8051 功能模块图（经 Intel 公司许可使用）

市场始终需要一款低成本的微控制器。为了满足市场需求，Intel 推出了 i8051 的衍生版本，该版本在维系了 i8051 基本结构的基础上，保留了其中一个功能子集。为了进一步削减成本，引

脚由 i8051 的 40 个减少到 24 个，并采用塑料双列直插式封装。

## 2.9 RISC 简介

20世纪80年代早期，半导体工艺技术的发展使得基于RISC的微控制器设计成为可能。哈佛结构问世于20世纪50年代，由于它的成本较高，使得实施起来并不现实。随着20世纪80年代晶片上晶体管集成度的巨大飞跃，使得单芯片RISC微控制器设计有了长足发展。

RISC是以指令的交叠为基础的。多条指令并行执行，以维持每一个机器指令执行的有效吞吐量。当然，这需要处理器同软件的结合。RISC的执行需要优化软件编译器，以获得最大的指令执行率。

### 2.9.1 RISC 处理器

早期的RISC处理器由多个芯片组成，和i8080方式相同。总线将独立的RISC处理器、数据存储器和程序存储器连接在一起，然后RISC芯片通过系统总线连接到标准外围I/O设备。图2-18给出了一个采用MIPS R3000芯片组的典型系统模块表，其控制总线和存储总线与采用CISC体系的设备相同。

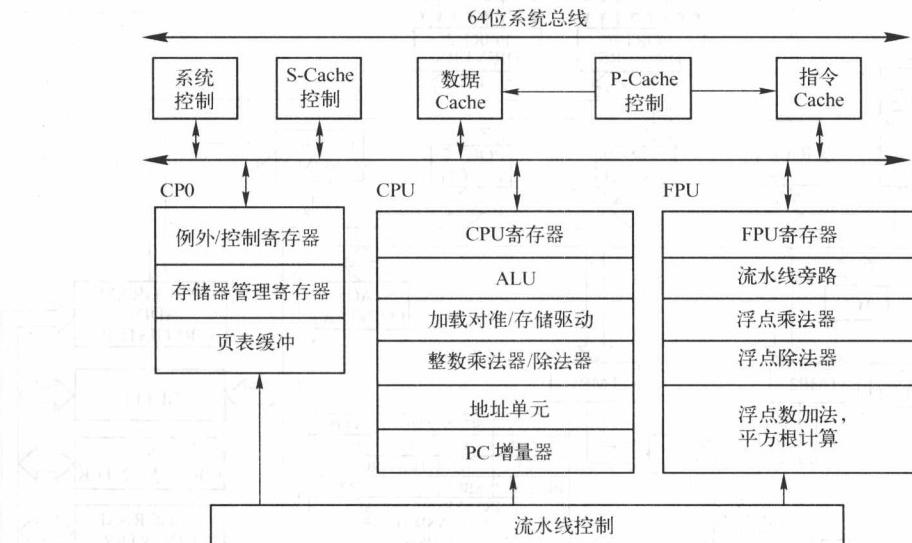


图 2-18 基于 RISC R4000 系统模块图（经 MIPS 科技股份有限公司许可使用）

20世纪70年代中期，第一块基于CISC体系的8位微控制器问世，比具有同样功能的RISC芯片组落后了近10年。到MIPS 2000 RISC芯片组上市时，它的性能已经超越了i80286。

### 2.9.2 RISC 的协同作用

直到20世纪80年代后期，RISC处理器才真正树立了其市场地位。这一时期，HP、SUN 和 MIPS 公司都推出了极富竞争力的产品，且与已有的 CISC 产品有了较大不同。但 RISC 芯片（MIPS 2000、HP-PA、Sun SPARC）并不是作为最终产品推向市场的，而 SUN、HP、MIPS 公司随即又推出了 RISC 工作站。图 2-19 列出了获得 MIPS RISC 授权的一些公司。

RISC处理器对于工作站设计来说更加有效，并且支持UNIX操作系统，支持C语言编程。这与采用Windows操作系统的CISC个人计算机有着极大的不同。RISC的这些特点就使其同CISC有了明确界限。

Actions Semi	ALi Corporation	AMD	ASUSTek
Atheros	Broadcom	Centillim	Chartered
Conexant Systems	ESS Technology	Infineon	IDT
LSI Logic	Marvell	Microchip	NEC
NXP	PMC-Sierra	Sony(SCEI)	Toshiba

图 2-19 MIPS RISC 授权公司列表

### 2.9.3 RISC 市场

MIPS 公司创立于 1982 年，该公司开创了一种独特的商业模式，从而为产品设计提供了另一种选择，它的出现对客户设计中半导体芯片的使用至关重要。由于有竞争才会有低价，因此客户们并不希望他们所需的产品只来源于一家公司。

半导体公司将他们的专利权授予其他厂商，而被授权公司则需要按合同规定交付一定的版税。

## 2.10 无晶圆半导体公司

MIPS 是作为无晶圆半导体公司成立的。他们并没有晶圆制造设备，然而产品却是由他们直接销售，看起来就像是一个半导体厂商。“无晶圆”的经营模式如图 2-20 所示。

MIPS 生产的第一款 RISC 处理芯片是 16 位的，称为 RS2000。它采用非易失性材料作为程序存储器，采用 RAM 作为数据存储器。这种设计最初是为 MIPS 工作站设计的。此外，MIPS 公司还在每一款 RISC 处理器中嵌入了 C 编译器。

为了进一步扩大市场份额，MIPS 公司将其芯片技术授予若干半导体厂商，他们之间达成了一个框架协议，从而允许半导体厂商将 MIPS 内核嵌入到他们的产品中。随后，一系列基于 RISC 体系结构的单芯片微控制器便涌入了市场。

### 2.10.1 RISC IP 核

这种新商业模式成为采用微处理器内核 IP 设计的前沿。MIPS 的授权公司很快就推出了各种基于同样内核技术的芯片。由于采用了基于同样 RISC 指令集体系的处理器，软件的通用性也增强了。

兼容软件是计算机领域的又一大进步，它使更多的芯片可以采用兼容的 C 编译器。此外，最重要的是允许了其他独资公司开发软硬件的调试工具。这些集成开发环境大大促进了产品的开发。

基于这种全新的商业模式，RISC 处理器在同传统 CISC 微处理器的竞争中一改往日的不利地位。企业如果采用相同的指令集体系进行产品设计，也就能够分享共同的开发环境。以下给出的一些通用的设计环境，请注意其应用的差异及广泛性。

- 物理库。
- 模拟混合信号。
- 算术逻辑功能。
- 测试内核。
- 外围设备内核。

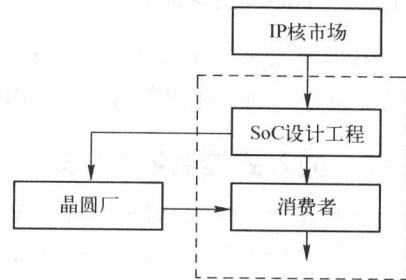


图 2-20 无晶圆经营模式

- 总线接口。
- 数字信号处理。
- 处理器和微控制器。
- 存储器单元。
- 安全、误差检测、调制。
- 视频、图片、音频。
- 有线通线。
- 无线通线。
- 平台级 IP 核。
- 硬件操作系统、实时操作系统。
- 核间通线。
- 嵌入式可配置逻辑。

CISC 处理器对市场的垄断就此打破，少数公司再也不能操控市场了，而芯片设计的限制也不复存在了。此时的市场又像从前一样充满着创造力，而 RISC 指令集体系结构成为了市场的主导力量。

半导体工艺限制了在单一晶片上集成的晶体管数量，这成为了 RISC 发展的一个限制因素。但设计概念已经在理论和试验电路上牢固建立起来，使得 LSI 工艺能够以经济的价格进行实际生产。

## 2.10.2 RISC 工艺流程

RISC 处理器与半导体工艺技术同步快速发展。随着处理器运算速度的提高，先前独立的芯片功能也集成到了处理器内核中，RISC 和 CISC 现在也遵循着同样的发展轨道。

## 2.11 嵌入式控制器 IP 核

20 世纪 80 年代，VLSI CAD (Very-Large-Scale integration Computer-aided design) 技术和半导体工艺技术的结合产生了一种全新的微控制器设计范例。工程师可以通过这种方法将微控制器的功能及更多的系统功能整合到单个芯片上，此芯片的微控制器部分就称为嵌入式 CPU 或者嵌入式内核（参见 2.24 节）。Infineon C166SV1 的功能模块图如图 2-21 所示。

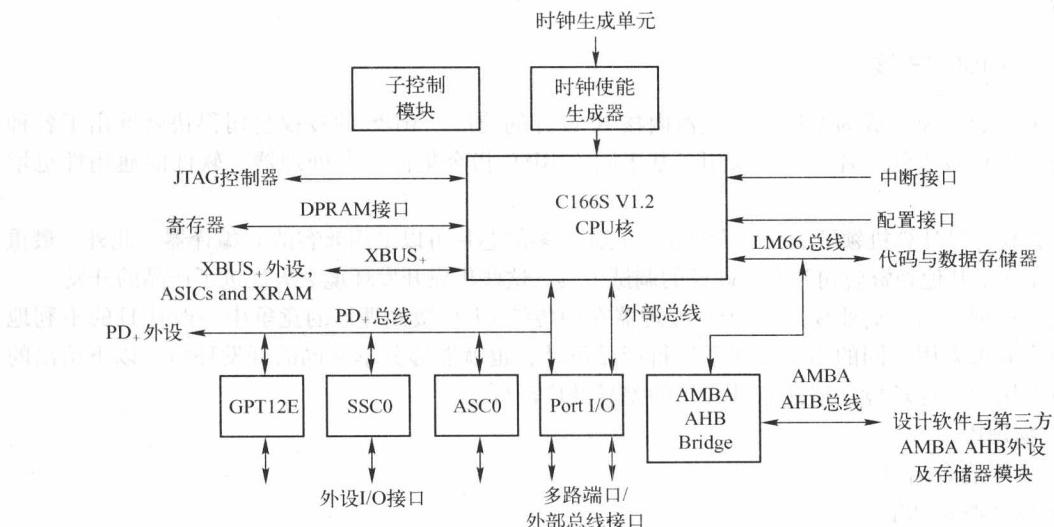


图 2-21 Infineon C166SV1 功能模块图（经英飞凌科技股份公司许可使用）

VLSI 设计工具将逻辑功能转换到晶片掩膜上，功能模块的硬件描述语言（Hardware Description Language, HDL）代码则被转换为数学模型。然后，通过软件对其进行处理，晶片的物理版图就产生了（见图 2-22）。

### 2.11.1 CISC IP 核

经 Intel 公司授权，设计公司可以使用 i8051 进行 SoC 设计，一旦该逻辑被转换成 HDL，它就可以整合到多种设计中。由于 i8051 IP 核采用了工业标准设计语言，所以它可以轻松地移植到第三方公司。

各行业熟练的开发工程师可以很方便地采用 i8051 进行 ASIC 和 SoC 设计，而且 i8051 的软件开发工具也并不昂贵。此外，i8051 的设计已经十分成熟，这在芯片设计中给了设计工程师们极大的信心。

市面上仍然有一些 CISC 微处理器，比如 Freescale 68HC11 和 Zilog Z8。然而，这些公司希望保护好他们的设计技术，因而他们并不对市场开放自己的指令集。结果，广泛使用的 CISC 指令集体系结构嵌入式处理器就只有 Intel i8051 了。

### 2.11.2 RISC IP 核

提供 RISC IP 核的公司越来越多，这对于“无晶圆”模式来说是一个十足的好消息。就 SoC 而言，企业可以独立开发出全定制微控制器。例如，ATMEL 公司就可以生产出一系列基于 ARM7 的芯片。

### 2.11.3 第三方 IP 核

直到 20 世纪 90 年代早期，另一种设计模式才得以确立，它将 IP 核整合到嵌入式处理器中。新的公司通常凭借基于 RISC 的设计占领新兴市场，ARM 公司就向市场中引进了又一种指令集体系结构。他们的商业模式是将其处理器内核的技术授权给其他公司而获得版税。

高性能的产品，比如路由器和服务器，需要最大化的计算处理能力。由于半导体技术的飞速发展（摩尔定律），在 SoC 上整合复杂但功能强大的 RISC 处理器已经成为可能，而市场上的 CISC 微控制器（Intel Pentium 和 AMD Athalon）却不能达到此目的。

整体而言，SoC 设计需要大量的投资。然而，通过权衡这种设计代价，SoC 工程师能够调整该设计需要投入的人力。嵌入式 RISC 核 SoC 确实需要投入巨大的设计精力，然而，它却在许多公司迅速传开，通过购买 IP 模块，工程师设计团队就可以像金融投资一样开发复杂的 SoC 了。

## 2.12 专用处理器

SoC 设计出现了另一种方式，即采用功能强大的 RISC IP 核。设计工程师只要从 MIPS 公司或 ARM 公司这样的厂商处获得 ISA IP 核，就可以专注于产品的功能设计。专用处理器（Application-Specific Processor, ASP）是为专门产品开发的，虽然产量相对较低，但由于使用的是已有的 IP 核，依然利润可观。

VLSI CAD 工具集的进步极大促进了处理器设计技术的发展。ISA 本身也可以依据不同应用

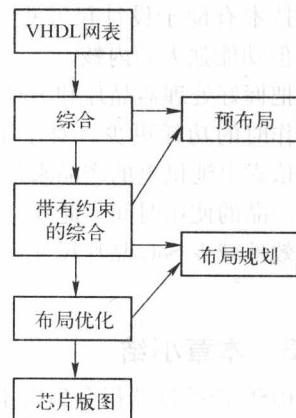


图 2-22 VLSI 设计流

进行配置。这种指令集和处理器内核的紧密集合进一步优化了处理器的性能，如图 2-23 所示。这些技术有助于设计尺寸更小（就晶片尺寸来说）但功能强大的内核。

把握好处理器晶片和 SoC 的尺寸比例，意味着工作时的功耗更少。对于手机或 MP3 播放器这类依靠电池供电的产品来说，减少耗电量就能增加产品的使用时间。此外，更小的内核封装能够有效地减少 SoC 晶片尺寸，这同样有利于降低成本。

## 2.13 本章小结

RISC 微控制器指令集由于应用的不同而向着不同方向发展。RISC 体系结构并没有受 CISC 体系结构太多影响，而且仍在不断进步中。现在 RISC 技术已经成为嵌入式微控制器和处理器设计中的推动力量。

现在，市场上的高端处理器将会成为下一代产品中的主流。设计工程师总是在寻求更强的处理能力，营销经理总是在寻求新产品，而 CEO 总是寻求更多利润。

RISC 的出现并没有受到现有市场的过多限制。许多半导体厂商现在已经能够触及处理器技术，他们可以开发出一系列基于 RISC 指令集体系结构的微控制器。

VLSI 技术的发展，使得单芯片上可以集成外围 I/O 功能，这样，芯片就拥有了类似 Intel i8051、Motorola 68HC12 或者 Zilog Z8 的功能。

## 习题

1. 什么推动了微控制器的发展？
2. 电子机械化设备的含义是什么？
3. 冯·诺依曼体系的基础是什么？
4. 请列出两种主流的指令集体系结构。
5. 简述小规模集成电路的定义。
6. 简述名词“总线”的定义。
7. 为什么 PCB 上要使用微控制器？
8. 什么是分区设计？
9. 请用图表列出 RS-232C 串行接口的基本信号。
10. 请描述微控制器和电子计算器的主要不同点。
11. 推动微控制器发展的最初动力是什么？
12. 十进制数 173 用二进制和十六进制如何表示？
13. 微控制器设计中最主要的系统约束条件是什么？
14. 为什么早期的微处理器需要多芯片才能运行？
15. 请列出微控制器的 5 个基本功能。
16. RISC 基于什么样的体系结构？
17. RISC 处理器最初应用于何处？
18. 简述“第二货源”的含义。
19. 什么“无晶圆”模式？
20. 什么是嵌入式内核技术的概念基础？
21. 什么是专用处理器？

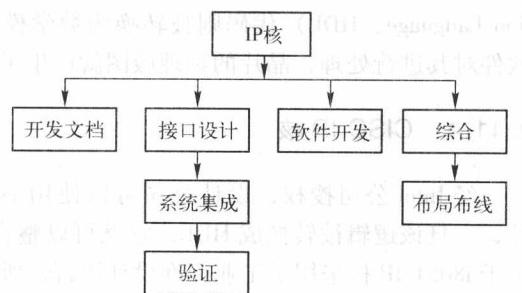


图 2-23 基于 IP 核的设计

## 参考文献

- C166S—A proven architecture takes on new shape. Technical presentation, Infineon Technologies, September 2002.
- Dadda, L. 1991. The evolution of computer architectures. *IEEE*: 9–16.
- 80C51FA/83C51FA Event-control CMOS single-chip 8-bit microcontroller data sheet. Intel Corporation.
- 8051 Microcontrollers hardware manual. 2005. ATMEL Corporation.
- Godfrey, M. D. and D. F. Hendry. 1993. The computers as von Neumann planned it. *IEEE Annals of the History of Computing* 15 (no.1): 1993 11–21.
- Miller, K. 2003. Development of the digital computer: Part 1. *IEEE Potentials*: 40–43.
- von Neumann, J. 1945. First draft of a report on the EDVAC. 1993. *IEEE Annals of the History of Computing* 15 (4): 27–75.

## 第3章 |

Embedded Microcontrollers and Processor Design

中文参考

# 嵌入式微控制器技术

- 本章目标：理解 RISC 处理器和 CISC 处理器的内部结构
- 主要内容：

  1. 半导体技术发展历程
  2. 设计抽象层次的概念
  3. 设计性能测试
  4. 软件、硬件和存储器技术
  5. 微代码层指令集设计概念
  6. RISC 处理器内部设计概念

## 3.0 集成电路

自从晶体管于 1949 年问世以来，半导体技术飞速发展，尤其是在一定空间里集成的晶体管数目不断增加，极大地推动了计算机设计的发展。

### 3.1 摩尔定律

1969 年，由 MOS 管（见图 3-1）构成的 256 位动态随机存取存储器（dynamic random access memory, DRAM）的面市，标志着半导体存储器时代的到来。随后，一部分厂商相继推出类似产品。到 20 世纪 70 年代，64 KB 的 DRAM 就已经开始大规模生产了。

戈登·摩尔是 Intel 公司的前总裁，前面内容中提到过，他得出了这样一个著名的公式：单个晶片上集成的晶体管数目每十八个月翻一番。对于微处理器来说，这就意味着可以实现更高的时钟频率。图 3-2 中给出了芯片时钟频率和单元密度的关系。

晶片上晶体管密度的增加不是简单地要求减小晶体管尺寸，而是要求设计技术和半导体工艺技术整体上的进步。除了晶体管尺寸的减小，半导体工艺技术也要确保能够生产出更大的晶片，以上两个方面的结合带来了晶体管密度的持续增长。

#### 3.1.1 微处理器的性能

DRAM 设计技术和半导体工艺技术是半导体革命的主要驱动力量，DRAM 中用来增

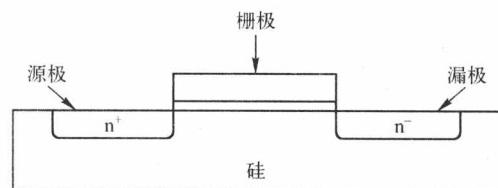


图 3-1 CMOS 管

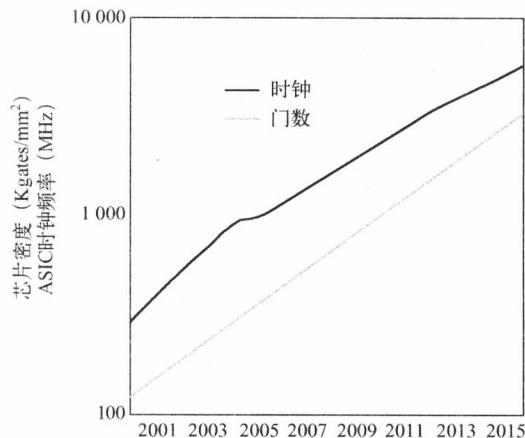


图 3-2 单元密度与时钟频率关系图  
(经 Tensilica 股份有限公司许可使用)

加晶体管密度的技术随后又应用到了微处理器和微控制器的设计之中。由于处理器中有效晶体管数量的增加，处理器的性能也就越来越强。

摩尔定律对于微处理器设计仍然适用。图 3-3 中给出了相对于不同晶体管密度下出现的几款主流微处理器。事实上，这不是简单地说工程师们可以采用更多的晶体管了，更重要的是他们可以采用更加成熟的处理器体系结构。由于半导体工艺技术上的进步，新一代处理器也呼之欲出。

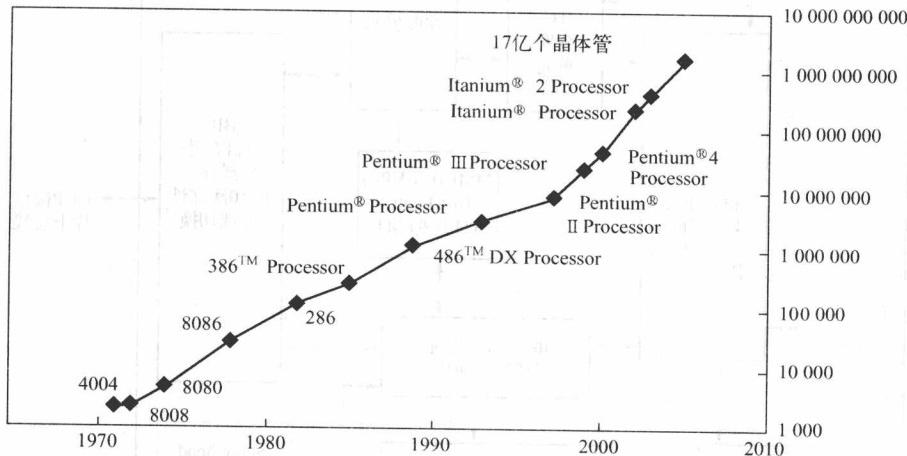


图 3-3 依据摩尔定律得到的微处理器晶体管数量 (经 Intel 公司许可使用)

正如摩尔定律所预测的，目前的技术显著地提高了微处理器的计算性能：频率从第一款 8 位 PC 的 4.77 MHz 达到 64 位 AMD Athalon 的 4 GHz。1970 年价值 \$ 20 000 000 的超级计算机的性能现在已被价值 \$ 500 的个人计算机赶上，性价比增长了 40 000 倍。

### 3.1.2 实现技术

DRAM 产品引领了全新的半导体工艺技术，此技术或将成为下一代处理器设计中的主流。同样，微处理器设计技术也带来了嵌入式微控制器设计的革新，如图 3-4 所示；因而可以说，微控制器是微处理器的衍生设计。

技术上的革新迎来了 64 位微控制器的问世。从前超级计算机才能达到的性能级别，现在可以整合到 SoC 上，这样就允许设计工程师嵌入各种处理器性能，以针对应用进行优化。

在任何时候进行工程设计，都在寻求能够充分利用现有技术。计算机设计工程师总是在现有技术的限制下谋求最优性能，最终采用的基本结构是设计技术和制造技术相互作用的结果。

冯·诺依曼基于当时可采用的技术，于 20 世纪 40 年代提出了他的设计理念：他采用了速度较慢的电子管技术；采用数据存储器，但存储容量有限；软件程序由硬件接线实现。

冯·诺依曼体系就是这三种主要技术综合的自然结果。其他的体系结构并没有得到广泛应用，不是因为它们还没出现，而是目前的技术还不能支持它们的应用。现在广泛采用的是普林斯顿体系结构（冯·诺依曼体系结构），因而现在要关注的是怎样优化其性能。

图 3-5 中的三个主要技术仍然是计算机体系结构中的重要部分。晶体管已经取代了电子管的地位，存储器也已扩大到兆字节级，软件功能也更加完善了。计算机系统要达到最佳的性能，需要综合考虑这三个因素，片面强调任一方面都会影响性能的发挥。

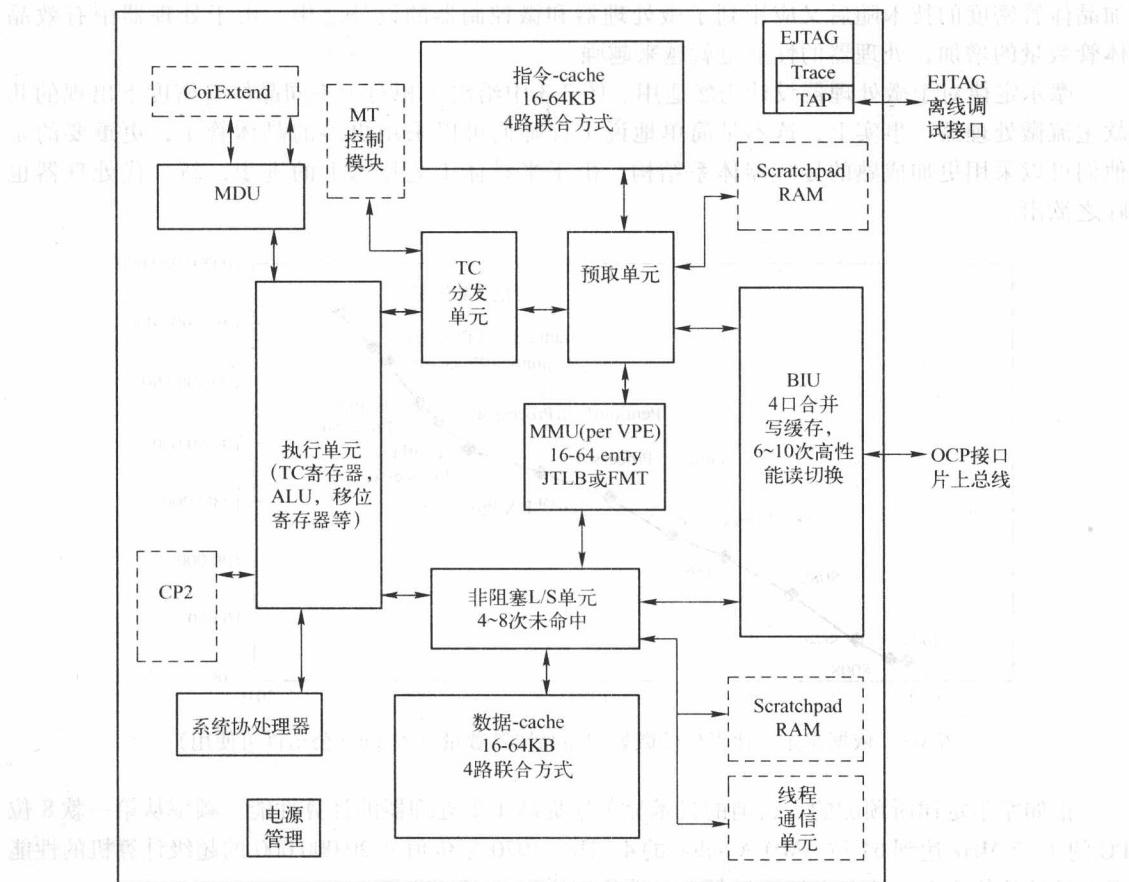


图 3-4 MIPS 34-Ke 32 位 CPU 内核模块图（经 MIPS 科技股份有限公司许可使用）

### 3.1.3 阿姆达尔定律

阿姆达尔定律告诉我们，要权衡相关技术以达到最优化性能是一件十分复杂的事情（见图 3-6）。系统由于采取某些优化措施所获得的性能提升正比于采取措施前的系统运行时间与采取措施后系统运行时间的比值，这意味着即使时钟频率翻一倍，吞吐量也不会相应地翻一倍。此外，存储器必须要有更快的指令和数据读取速度，软件程序结构需要支持连续的指令执行。



图 3-5 三种主要的计算机技术

$$\begin{aligned} \text{ExTime}_{\text{new}} &= \text{ExTime}_{\text{old}} \times (0.9 * 1.0 + 1 * 0.5) \\ &= 0.95 \times \text{ExTime}_{\text{old}} \\ \text{Speedup}_{\text{overall}} &= \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{0.95} = 1.053 \\ &<<2 \end{aligned}$$

图 3-6 阿姆达尔定律

技术一直都在进步着。由于新技术不断涌现，设计工程师可以将这些新技术整合到他们的产品中。然而，新技术的采用也常受到市场的制约，经济因素在设计中扮演着极为重要的角色。目前市场上出现的微控制器可以说是市场制约下的技术成就。

### 3.1.4 技术融合

为此，处理器体系结构的发展可能停滞不前，于是更新的结构技术可能并不会很快进入市

场，基于 Intel 处理器结构的个人电脑就是这样的典型案例。由于软件受限于固定指令集，因而只能寻求逻辑和存储技术方面的进步。这些软件技术方面的约束可能导致产品不能达到原本可以达到的最佳性能。

一些应用程序并不要求很高的软件兼容性，这样就大大降低了处理器设计工程师的工作难度，使他们能够采用其他的体系结构。早在 20 世纪 80 年代，硬件技术和存储器技术就已经达到此水平。继普林斯顿结构之后，处理器、存储器和软件技术的这种融合使得哈佛结构已经能够实现。

## 3.2 设计抽象

在有了处理器设计的概念时，就有了软件指令集的定义，但是它并不是硬件设计的产物，而是硬件设计的驱动力量。设计工程师需要在以上 3 个技术方面做出权衡，以获得最优性能。指令集体系结构（ISA）就是该优化过程的一个基础部分。

一旦确定了指令集体系结构，就可以进行处理器设计。处理器设计的基本目标就是达到最快的指令执行速度。处理器逻辑依据指令执行的需要进行设计；存储器结构设计旨在获得最大的指令和数据读取速率；软件程序则应当优化为采用最少的指令。

指令集体系结构是计算机系统物理设计的一个抽象概念。它包括各种可能的设计实现。与其说这是特定的设计标准，不如说这是一种设计理念。指令集体系结构同样意味着将处理器作为一个系统来进行逻辑设计，而不仅仅是一个指令的集合。图 3-7 描绘了指令集体系结构在软件、硬件抽象层之间的位置。

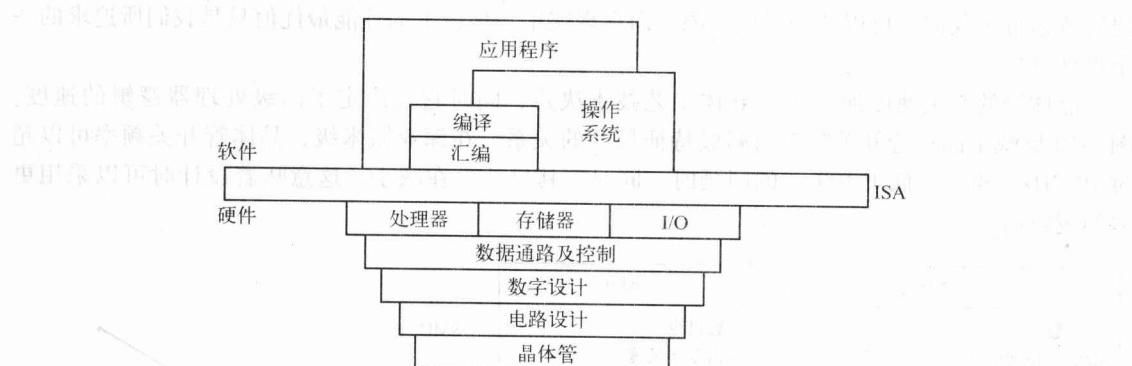


图 3-7 抽象层

微处理器体系结构分为 CISC 和 RISC 两大类，它们之间并没有严格的划分，但是采用不同体系结构设计的处理器会呈现出不同的特点。

### 3.2.1 指令集体系结构

目前存在很多重要的指令集体系结构，图 3-8 中列出了一些常见的指令集体系结构。定义指令集体系结构主要用来解决面向应用设计中遇到的问题，现实中的指令集体系结构通过增加性价比来克服执行成本方面的不足。

某种程度上，处理器体系结构通常会相互融合各自的特征。自上往下，CISC 和 RISC 有诸多共同点；自下往上，它们的设计理念却截然不同。然而，随着处理器基本逻辑、存储器和软件技术的发

CISC	复杂指令集体系结构
RISC	精简指令集体系结构
VLIW	超长指令字
EPIC	增强并行指令计算机
MISC	多发射指令集计算机
DSP	数字信号处理
GPU	图像处理

图 3-8 指令集体系结构

展, CISC 和 RISC 的相互作用显得更为有益。采用合理使能技术的新设计可以有效地整合 CISC 和 RISC, 进一步地, 这种“混合”体系结构又带来了更佳的性能。

### 3.2.2 处理器家族

指令集体系结构可以定义为处理器家族中的一个分支。图 3-8 按照处理器正式上市的时间顺序进行编排, 同时还指出, 当技术累积到一定时间后, 便能够推出新的指令集体系结构。

从图中还可以看出传统的 CISC 和 RISC 终将会让位于新的指令集体系结构, 这也反映了处理器逻辑、存储器和软件技术的持续发展。然而, 未来在商用领域里能否成功并不能预知, 因为一种指令集体系结构能否从理论变成现实取决于它能否带来利润。

## 3.3 RISC 和 CISC

总体上说, CISC 和 RISC 在设计趋势上是可以做个比较的。图 3-9 所示的表中给出了一个简单的对比, 该表比较粗略, 还有许多方面的差异没有给出。

### 3.3.1 处理器技术

在理想计算机中, 程序按照设计预想的速度运行。事实上要达到这个目标是非常复杂的。处理器通过执行指令来完成所需的操作, 而数据的处理则与输入/输出同步进行。一旦所有功能模块协调运行, 我们就可以获得最优性能。而在现实中, 理论上的性能最优值只是我们所追求的一个设计目标。

晶体管的开关速度通常由半导体工艺技术决定, 同时它又决定了门级处理器逻辑的速度。图 3-10 反映了晶体管开关频率与漏极特征尺寸的关系。在深亚微米级, 晶体管开关频率可以超过 10 GHz, 相当于低于 100 ps 的门延时。同时, 其尺寸也在减小, 这意味着设计时可以采用更多的逻辑门。

CISC	RISC
指令数目多	指令数目少
指令字长度可变	固定指令字长度
寻址方式较多	寻址方式较少
基于存储器的数据操作	基于寄存器的数据操作
专用寄存器多	正交寄存器组
指令执行于多个微指令周期	单周期指令
汇编语言驱动	高级语言驱动

图 3-9 RISC 与 CISC 对照表

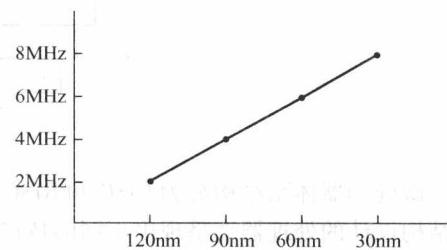


图 3-10 晶体管开关频率

### 3.3.2 性能评估

在任何一个抽象级别对计算机系统进行性能评估都是相对的。它就像对量子的估计, 某种程度上说的越精确, 事实上反而越不精确。但是, 设计工程师要在计算机结构上做出准确的决定, 需要获得一些性能评估数据。

简单地说, 指令执行得越快, 数据处理得就越多。这是数据处理的一个基本理论。接下来的问题就是找到一个性能评估的尺度, 或者说是一个基准, 用以确定处理器的设计效率。

$$\frac{\text{Time}}{\text{Program}} = \text{Time}_{\text{Program}} = \text{Instructions}_{\text{Program}} * \text{Cycles}_{\text{Instruction}} * \text{Time}_{\text{Cycle}}$$

处理器设计时，每周期的时间 ( $\text{Time}_{\text{Cycle}}$ ) 已经确定，即为处理器逻辑的时钟频率。减少程序中指令的数目 ( $\text{Instructions}_{\text{Program}}$ ) 不仅能够提高性能，而且也能够简化程序。这样一来，程序执行效率和系统性能都能得到提升。

程序执行所需时间可以通过三种方式来削减：减少程序代码中使用指令的数量；减少指令的执行周期；降低每个周期的时间。

### 3.3.3 程序指令

现假设有一个程序使用了最少的指令，这样就确定了上述方程的第一项，而硬件由设计技术确定。这样，唯一能够用来提高性能（降低程序执行时间）的方法就是减少指令数量，这就推动了 CISC 设计的发展。

方程式中的三项以递归的方式相互作用，增加指令数量，会相应增加处理器的复杂程度，导致指令执行周期更长。这些因素都要好好权衡才能做出最好的决定。阿姆达尔定律量化了性能的描述指标，便于更好地做出最佳选择。

在现实应用中，系统级的性能评价指标不仅仅是简单的指令和处理器周期。有一套基准程序可以用来进行处理器评价。尽管如此，目前仍然没有在单一基准上形成一致。

### 3.3.4 指令成本

每行代码的成本是与代码内容相互独立的，它与程序员的编程效率直接相关。假设一个程序员每天编写 40 行带有完整文档的调试代码，每条指令的成本就可以非常容易地得出，即程序员所需总成本除以指令数。例如，雇佣一个程序员在一个工作日要花费 \$800，一天可以完成带有完整文档的测试代码 40 行，每条指令的成本粗略算起来就是 \$20。

$$\text{Time}_{\text{inst}} - \text{Cost} = \text{Programmer Cost} \cdot \text{Day} / \#_{\text{Instructions}} = \$800 / 40 \cong \$20$$

正如我们所见，减少指令数量可以提高处理器性能，同时也减少了程序的内存占用量，这就意味着同时也节省了内存资源，如图 3-11 所示。减少指令数量，既降低了程序的成本，也提高了程序性能，同时又降低了硬件（内存）开销。这些优点都成为 CISC 技术发展的强大推动力。

### 3.3.5 微代码指令

增加指令数量会相应地增加处理器逻辑的复杂度。从阿姆达尔等式可以看出，如果指令数量超过一定点，反而可能降低收益。因而需要另一种方式以达到有效处理更多指令的目的，这就是微代码指令技术。

在处理器内部，机器指令的执行分为几个步骤。这样，指令在处理器内核逻辑的执行过程中就像一个小程序，增加新指令仅仅需要增加一些微代码指令。

很少有指令能够明显减小代码大小。但是，存储器通常价格较高，需要我们合理利用，同时又要在相同的地址空间存储更大、更多的程序模块。这些特点引领了指令集体系结构扩展技术的发展。典型例子就是 Intel Pentium 处理器的 MMX™ 技术。这些指令不仅提高了图像处理能力，也有效地简化了编程软件。

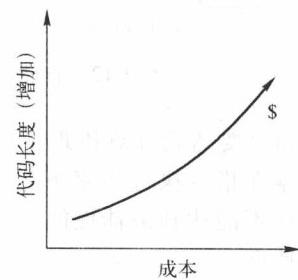


图 3-11 程序大小与成本关系

这是 CISC 指令集体系结构处理器设计的重要设计思路。复杂指令是相对于简单指令来说的，是比较适用的一种指令集体系结构。对上述三种主要技术来说，这意味着更快的执行速度、更少的内存占用量和更低的程序开发成本。

机器指令对于程序员来说是可见的，而微代码却是不可见的。微代码指令的执行顺序由处理器设计者决定，它们被嵌入到内核逻辑中。从某种意义上说，CISC 处理器包含嵌入式微控制器。例如，Pentium 处理器就可以看作针对通用计算机应用的专用处理器。

从这些简化了的性能评估指标来看，我们会发现上述三种基本技术之间的相互作用。硬件设计致力于让指令执行更快，内存占用更少。这些不仅在高层软件级如此，在微代码级同样适用。

### 3.4 存储器技术

分级存储器中数据的大小和访问速度并不一致，这一矛盾将指令集体系结构的发展引向深入。相对来说，访问速度较快的存储器价格会更高一些。从某种程度上说，指令越能有效利用存储器，处理器的性能就越好，系统成本也越低。

预存储程序的计算机需要存储器预存储指令和数据。事实上，将存储器整合到处理器逻辑中并不实际，也不可取。晶片上的晶体管数量是有限的，如果将它们用到处理器逻辑中就会更有价值。

通常对于存储器有三种基本要求：快，更快，最快。程序员对存储器的要求或许就是：大，更大，最大。如果不计速度和容量，处理器执行指令不可能快于指令从存储器取出的时间，如图 3-12 所示。操作数据时也不能比数据的存储更快。事实上，在处理器设计过程中存在着一些限制因素。

由摩尔定律我们知道，存储器容量（单个晶片）每 18 个月翻一番。目前存储器访问时间已经有所降低，如图 3-13 所示，但是这仍然不能与处理器频率相提并论。虽然 DRAM 相对较快，但是仍比处理器至少慢十倍。

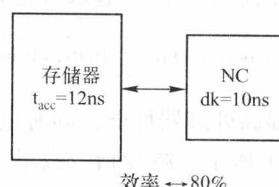
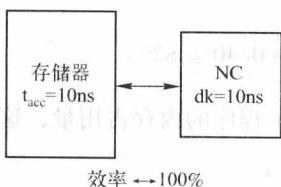


图 3-12 存储器处理器接口速度

DRAM 类型	频率 (MHz)
PC-3200	400
PC-5300	533
PC-6400	800
PC-8000	1000

图 3-13 PC 存储器速度

如果要达到计算机理论最佳性能 (Computer's Theoretical Performance, CTP)，需要指令和数据在指令执行频率下是可用的。如果指令或数据不可用，那么处理器就会延时执行，这样就不能达到最佳性能。如果从主存直接执行指令和数据，则能够尽可能地匹配上处理器的速度。

#### 3.4.1 局部性

对软件的程序流程的研究表明，程序的执行大体上集中在一些小范围的指令群中。典型例子就是子程序或者数据组的循环，如图 3-14 所示。程序代码将会执行某区域的一些指令，然后分支到其他新的区域；在新地址处，程序同样也会执行一系列指令。这个过程称为局部性原理，程序和数据组的执行都涉及这个概念。

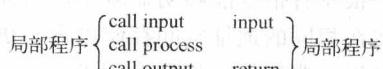


图 3-14 程序局部性

我们必须注意到，指令和数据在执行中都要是可用的。但是，它们的属性却大不相同。虽然它们都以流的形式进行处理，但是指令执行必须预先确定顺序。尽管数据也展现出局部性特点，但并没有固有的处理顺序。

现实中的应用对存储器有不同的要求，有些应用需要大而复杂的程序来处理相对较小数量的数据（字处理），另一个极端就是很小的程序要处理大批的数据（网络搜索引擎）。

在只有单个存储器的冯·诺依曼体系结构的 CISC 计算机中，从存储器中取指令和数据面临着同样的挑战。被修改的数据需要写回到存储器中，处理器需要以最快的速度写回数据以进行下一次数据处理。

在任何时候进行设计，设计工程师都只采用目前可用的技术。随着计算机设计技术的发展，许多新技术应用到计算机设计中以让处理器保持 busy 状态。确定处理器 busy 状态的方式多种多样，总体上说就是让处理器的功能单元尽可能处于工作状态。

### 3.4.2 存储器分级

存储器的分级概念是基于局部性原理的。处理器执行单元和主存之间插入了一个缓冲区或者高速缓存，处理器逻辑的取指部分可以直接从高速缓存中取指，这样延时就大大降低了，数据也从高速缓存直接传输到寄存器堆中。这样，处理器就可以以最快的方式取到指令，也就越接近理论最佳性能。

通用存储器的分级结构已经产生了。目前定义了四种抽象级的存储器——寄存器、高速缓存、主存和文件存储器，如图 3-15 所示。越接近处理器的存储器，其访问速度越快。由于各级存储器的设计目标不同，其大小和性能也不尽相同。对于微控制器设计来说，可能只采用一级存储器，也可能都会采用。

各级存储器的访问速度差别很大。一般地，存储空间越大的存储器访问速度越慢。图 3-16 给出了一个四级存储器的相对访问速度，由于文件存储器是电动机械实现的（硬盘），因此速度明显慢于半导体存储器。

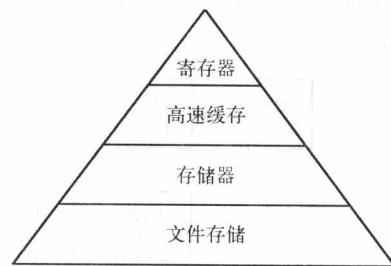


图 3-15 存储器分级

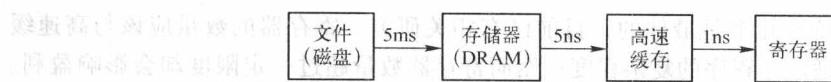


图 3-16 存储器分级访问时间

正是由于局部性原理，存储器分级技术才能正常进行。在文件级，这可以应用到复杂程序的实现中，例如网络路由选择。在各级存储器间附加的高速缓存（缓冲区）可以进一步降低访问时间延时。硬盘和 CD 驱动都包括用来提高性能的高速缓存以连接到主存。只要处理器在最短的时间内接收到所需的指令和数据，存储器分级的目标就算达到了。

### 3.4.3 高速缓存

正如我们所见，存储器分级中最快的是高速缓存。而正如摩尔定律所描述的，主存的规模仍然在不断地翻番。2G 的内存将会被 4G 所取代，而随着 DRAM 技术的发展，8G 也将登场。

另一方面，快速存储设备也在不断发展。越来越简化的设计规则提高了单个晶片上所含晶体管的数量。此外，片上高速缓存所使用的晶体管数量也在不断增加，于是高速缓存的容量增大

了，处理器的性能也在相应提高。

现在，要在单个晶片上集成 25 000 000 个晶体管已经不是难事。基于此，不仅寄存器可以整合到处理器晶片上，高速缓存也可以整合进去。这样，处理器似乎有了一个专门的指令和数据存储器。

### 3.4.4 一级缓存和二级缓存

目前，高速缓存本身也已经可以分割。最快的高速缓存作为一级缓存（L1），二级缓存（L2）容量稍大些，位于另一个晶片上，但是仍在同一个芯片封装中（例如 Pentium），如图 3-17 所示。

### 3.4.5 数据寄存器

寄存器是位于处理器内部的存储器，其中存储的数据可以在处理器执行周期中进行读写操作。如果寄存器中的数据是可用的，只要指令需要，就可以直接操作。一旦数据可以在高速缓存和寄存器堆中快速传输，处理器便可达到最佳性能。

寄存器可以通过两种途径寻址。它们可以定义为特定的地址，隐含在指令代码中。另外，地址可以存放在存储器中，如图 3-18 所示。针对不同的应用，两种寻址方式都可以提供很好的性能。

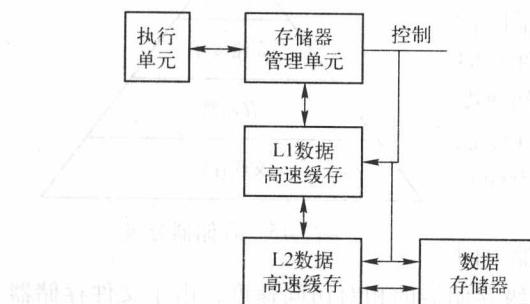


图 3-17 带有 L1 和 L2 高速缓存的处理器

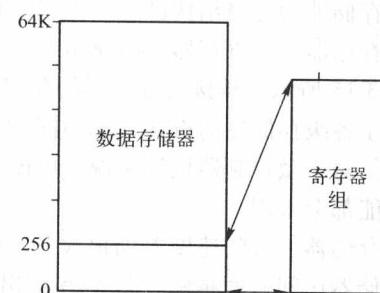


图 3-18 存储器映射寄存器

至于寄存器的数量维持在几个是最佳的，目前已有相关研究。寄存器的数量应该与高速缓存的大小相适应，而不是适应于程序的复杂程度；然而寄存器数量超过一定限度却会影响盈利。这些因素共同决定了寄存器数量和高速缓存的大小。

### 3.4.6 指令队列

对于处理器设计工程师来说，处理指令在处理器内核逻辑的进出是一个巨大的挑战。工程师需要确保软件代码定义的指令队列在处理器处理时是有效的，这个过程与存储器的读写操作大不相同。

指令不可能仅仅在处理器的存储器间移动并随机地执行。目前仍然没有相应的指令存储器的概念，指令只是暂存在队列之中，队列中的指令依次进入存储器，如图 3-19 所示。

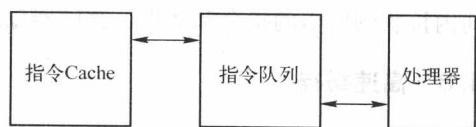


图 3-19 指令队列

### 3.4.7 分支指令

一旦遇上分支指令，指令就不会按原指令顺序继续进行。分支指令有很多种类型，如图 3-20

所示。不论是哪种类型的分支指令，都会打断原来指令的顺序进行，而强制处理器从一个新的位置取指。这可能延迟指令的执行，直到有新指令取指完成为止。

分支指令类型	代码
GOTO指令	Goto Skip
条件指令	If x=5 then goto Skip
CALL(n)调用指令	Call Raster
RETURN返回指令	Return to(Call Address +1)
RETFIE中断返回指令	Return from Interrupt Enable(n+1)
BRA分支指令	Branch Skip

图 3-20 MicroChip PIC 18F4520 中的分支类型指令

由于局部性原理，分支指令的地址不是完全随机的。如果队列还有足够的存储空间，那么分支指令就可能已经取到队列之中了。这种情况下，指令执行可能就不会延时，处理器将会继续高速执行运算。

程序指令执行延时的情况有两个典型例子：某条指令并不是顺序执行的；寄存器中的数据不可用。这两个问题可以通过处理器分级来得到有效解决。

### 3.4.8 存储器访问延迟

从硬盘直接传输一条指令或者数据到处理器是一个很慢的过程，这个时间包含高速缓存的访问时间、主存访问时间和文件存储器访问时间。文件存储器是电动机械的，访问速度最慢，因而以上时间主要取决于文件存储器的访问时间。通常，存储器访问延迟满足以下公式：

$$T_{\text{access}} = t_{\text{cache}} + t_{\text{mem}} + t_{\text{file}} = 2\text{ns} + 8\text{ns} + 7\text{ms} = 7.00001\text{ms}$$

在大多数程序中，不到 5% 的代码可能随时执行，例如电子表格程序中只有少数功能会随时用到。这样，只有小部分指令需要长时间存放在主存中。

在实际应用中，指令不可能逐一地在各级存储器中传输，这样平均延时太长。通常，邻近的存储区域被划分为存储器块，在不同的应用中，块的大小从 1K 到 4K 不等。如果某指令不在高速缓存中，那么包含该指令的块将整个调到高速缓存中。

指令延迟是一个十分关键的定义。不同应用中的指令延迟不同。如果指令在程序需要它是可用的，那么就应用而言，这些指令总是可用的。而对于处理器来说，并不会认为存储器存在分级。从应用的角度来说，只存在一个大容量且高速的存储器。

### 3.4.9 高速缓存模块

数据必须存储在存储设备中。如果高速缓存中的某个模块已经被修改，那么它也必须同样写回到主存中去。在高速缓存的设计中有一个双向的数据传输，然而，只有已被修改的模块需要写回到主存中去，这样就将高速缓存的写回操作降低到了最少。

各级存储器间存储块传输的重要性再强调也不过分。在任何含有高速缓存的处理器设计中，这都是系统性能的门控因子。对于微控制器来说，指令和数据流必须要高度协调。实时应用中，处理器必须在合适的时候处理合适的指令，以在事件发生时及时处理事件。

管理基于高速缓存的设计，需要一些附加逻辑，然而这又提高了设计的复杂度和芯片的成本。此外，增加片上高速缓存也会增加晶片的尺寸，且会直接提高工艺成本。无缓存的微控制器成本会稍低，但是同时也牺牲了一些性能。在 Microchip PIC 系列中，所有片上存储器都以寄存

器的方式存在，这就排除了高速缓存会遇到的问题，如图 3-21 所示。

地址	寄存器名	地址	寄存器名	地址	寄存器名	地址	寄存器名
FFFh	TOSU	FDFh	INDF2 <sup>(1)</sup>	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2 <sup>(1)</sup>	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2 <sup>(1)</sup>	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2 <sup>(1)</sup>	FBCh	CCPR2H	F9Ch	__(2)
FFBh	PCLATU	FDBh	PLUSW2 <sup>(1)</sup>	FBBh	CCPR2L	F9Bh	OSCTUNE
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	__(2)
FF9h	PCL	FD9h	FSR2L	FB9h	__(2)	F90h	__(2)
FF8h	TBLPTRU	FD8h	STATUS	FB8h	BAUDCON	F98h	__(2)
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	PWM1CON <sup>(3)</sup>	F97h	__(2)
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	ECCP1AS <sup>(3)</sup>	F96h	TRISE <sup>(3)</sup>
FF5h	TABLAT	FD5h	T0CON	FB5h	CVRCON	F95h	TRISD <sup>(3)</sup>
FF4h	PRODH	FD4h	__(2)	FB4h	CMCON	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	HLVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	__(2)
FF0h	INTCON3	FD0h	RCON	FB0h	SPBRGH	F90h	__(2)
FEFh	INDF0 <sup>(1)</sup>	FCFh	TMR1H	FAFh	SPBRG	F8Fh	__(2)
FEEh	POSTINC0 <sup>(1)</sup>	FCEh	TMR1L	FAEh	RCREG	F8Eh	__(2)
FEDh	POSTDEC0 <sup>(1)</sup>	FCDh	T1CON	FADh	TXREG	F8Dh	LATE <sup>(3)</sup>
FECh	PREINC0 <sup>(1)</sup>	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD <sup>(3)</sup>
FEBh	PLUSW0 <sup>(1)</sup>	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	__(2)	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	__(2)
FE7h	INDF1 <sup>(1)</sup>	FC7h	SSPSTAT	FA7h	EECON2 <sup>(1)</sup>	F87h	__(2)
FE6h	POSTINC1 <sup>(1)</sup>	FC6h	SSPCON1	FA6h	EECON1	F86h	__(2)
FE5h	POSTDEC1 <sup>(1)</sup>	FC5h	SSPCON2	FA5h	__(2)	F85h	__(2)
FE4h	PREINC1 <sup>(1)</sup>	FC4h	ADRESH	FA4h	__(2)	F84h	PORTE <sup>(3)</sup>
FE3h	PLUSW1 <sup>(1)</sup>	FC3h	ADRESL	FA3h	__(2)	F83h	PORTD <sup>(3)</sup>
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIA2	F81h	PORTB
FE0h	BSR	FC0h	ADCON2	FA0h	PIE2	F80h	PORTA

图 3-21 PIC 18F4520 中的寄存器映射（经微芯科技股份有限公司许可使用）

高性能不可避免地带来高成本，但也带来了更强大的功能。我们所追求的当然是以最低成本谋求最佳性能，这在当前高科技世界中就是成功与失败的差别。

### 3.5 指令处理

处理器本身不过是一个复杂的逻辑功能设计。汇编指令会转化为更基础的机器指令，通常，单个汇编指令需要几个机器指令（微码）来实现。采用二进制逻辑的 0 和 1 就能够满足处理器门级的逻辑要求，如图 3-22 所示。

即使写一个非常短的机器指令程序，也是一件十分枯燥的事情，因为它的每一个指令直接用二进制来编码，分支指令 GOTO 的分支地址也需要用二进制表示。用机器语言编写的程序，既难以阅读，也容易出错。最重要的是，即使开发一个极短的程序，也需要花费大量的软件工程

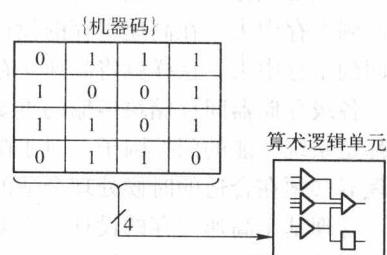


图 3-22 指令机器码

资源。计算机硬件的不断更新换代，使寄存器的位数越来越多，因此对汇编语言提出了更高的要求。

### 3.5.1 汇编语言

计算机性能的提高以及存储器大小的增加推进了软件开发技术的发展。汇编编译器就是专门为将符号代码转化为机器指令而开发的。例如，计算机中产生加指令的结构表示为 ADD。

汇编语言指令更便于程序员使用，它的符号通常是相应符号单词的缩写，加即为 ADD，乘为 MUL，等等，如图 3-23 所示；每个寄存器都有其标志名称，如 R1、R2 和 R3；数据变量同样也由符号表示，而不是二进制码。

助记符，操作数	描述	周期数	16位指令字			
			MSB			LSB
<b>位操作指令</b>						
BCF f,b,a	Bit Clear f	1	1001	bbba	ffff	ffff
BSF f,b,a	Bit Set f	1	1000	bbba	ffff	ffff
BTFS C f,b,a	Bit Test f,Skip if Clear	1(2 or 3)	1011	bbba	ffff	ffff
BTFS S f,b,a	Bit Test f,Skip if set	1(2 or 3)	1010	bbba	ffff	ffff
BTG f,b,a	Bit Toggle f	1	0111	bbba	ffff	ffff

图 3-23 PIC 18F4520 汇编指令（经微芯科技股份有限公司许可使用）

汇编语言大大增强了程序员写代码的能力。写代码不仅指生成指令，而且也包含程序的调试。汇编语言同时也提高了程序员的编程效率，它可以说是增强计算机性能和实用性的关键技术之一。

### 3.5.2 程序编译器

软件程序是由程序员一行一行写成的。开发语言的抽象级别越高，程序员的开发效率就越高。编译器可以将高层语言描述的程序转换为机器指令序列。程序员使用 C 语言之类的高层语言可以方便地编写更多程序，C 编译器可以大大提高产生代码的效率，例如以下程序：

```
void ee_byte(byte x)
{
    int i;
    i = 0x80;
    do
    {
        SCL = 0;
        TRIS1 = 0;
        ee_delay();
        if (x&i) SDA = 1;
        else SDA = 0;
        ee_delay();
        SCL = 1;
        ee_delay();
        i >>= 1;
    }while(i != 0);
}
```

采用编译器可以大大提高编程效率，但是正如我们所见，这又带来了更高的成本，并且不能

达到最佳性能。在 CISC 指令集体系结构中，编译器产生的代码会更大，而且并不是最佳的。这是因为不能很好地操控专用寄存器堆。在微控制器设计中，所有能用到的技术都要为获得最佳性能服务，汇编程序编程就是一种强制性的关键代码优化。

### 3.5.3 硬编码指令

一个项目的软件开发成本往往高于硬件设计成本。可以证明，指令数量的减少可以提高系统的性能。指令的最佳数目其实是一个相对的定义，通常被认为是达到最佳性能所需指令的最少数目。

程序员编写更少的指令，就能减少代码的成本，所生成的程序所占内存也会减少，这样就可以适当考虑采用价格稍贵但速度快的存储器了。

## 3.6 程序设计

CISC 处理器是一款通用处理器，其指令是为合理处理各种应用程序而开发的。为了满足程序员对设计效率的需求，CISC 又定义了一些附加指令。这些指令是专门针对应用的，就是说这些指令是为了让处理器应对各种特定问题，如数据处理，而专门定义的。

新指令会随着新应用的产生而产生。这也会对寻求更加复杂的处理器逻辑与新指令的固有特性之间的平衡形成压力，包含硬布线逻辑的处理器设计更加复杂。

正如我们所见，编译器可以提高程序员的编程效率，但是这是以系统性能的降低为代价的。为了获得最佳性能，软件程序员会采用汇编代码编写关键例程，如图 3-24 所示。对于基于微控制器的系统来说也是这样，为了获取最佳性能，采用汇编语言是唯一方法。这就是阿姆达尔定律的直接结论。

### 3.6.1 程序代码大小变化

微控制器设计的另一个关键因素是代码大小。随着硬件功能的增强，更复杂的应用也已可以实现。这就意味着为了提高性能，可以采用规模更大的程序。然而，快速存储设备的价格正变得越来越低，容量却越来越大，内存成本和软件代码大小的比率仍然维持稳定。内存的可用性通常决定了大程序运行的可能性，反过来大程序也增加了对大内存的需求。

### 3.6.2 CISC 指令集

一个基于 CISC 指令集体系结构的处理器可能有数百个可能的指令组合。1998 年问世的 Intel Pentium IV 有超过 200 个的操作码。支持如此多种类的指令，需要复杂的微码引擎，而执行复杂指令的微循环的数量则会限制处理器的运行速度。

随着一级缓存和二级缓存概念的问世（见图 3-25），存储器的速度相对来说已经能够匹配处理器的速度，这部分性能也已经通过半导体技术的发展得以实现。这样一来，人们就开始关注软件技术的发展了。目前的指令越来越复杂，寄存器的数量越来越多，高速缓存间的数据传递也在增多，这些都成为制约发展的瓶颈。

除了前文提到的三个主要因素之外，还有一个十分重要的因素影响着最佳性能的实现。目前的 PC 市场广泛应用 Intel 指令集体系结构，软件则基于微软的视窗操作系统，这成为 PC 市场

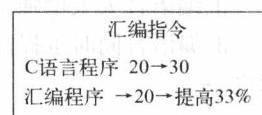


图 3-24 高层语言效率

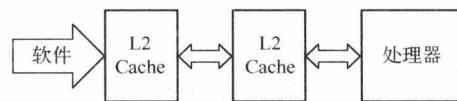


图 3-25 高速缓存冲突

事实上的标准。因此，指令集体系结构的基础结构变化并不会发生，性能提升只是有限的增量变化。这反过来又限制了计算机达到上述三种主要技术所允许的最佳性能。

### 3.7 统一指令集

高层编程语言的编程效率越来越高，而对编程效率不断增长的需求又推动了编译器技术的发展。

CISC 编译器在处理含有固定功能寄存器的寄存器堆时依然很困难。程序员可以使用汇编来操作 CISC 寄存器堆。指令的复杂度有一个峰值，超过这个峰值就可能降低受益。例如 Intel Pentium 微处理器和 AMD Athalon 微处理器，增加处理器性能的关键因素是加快发展最基础的半导体工艺技术。

体系结构的加强推动了工艺技术的发展。在市场的引导下，软件技术围绕在一些固定数量的指令上。不像早期 CISC 的发展那样，为了实际应用而开发新指令是受到禁止的。

#### 3.7.1 工业标准软件

Intel 引入了新的指令集体系结构，而 i8086 则成了事实上的工业标准，未来的指令集体系结构固定在了那个时间点。事实上，加强处理器逻辑的性能是可能的，但是基本指令必须按照与传统处理器兼容的方式运行。

指令集也可能成为设计工程师在设计过程中的限制因素，要强性能可以依靠软件、存储器和半导体技术的发展，但是必须在已有的指令集范围内。这限制了芯片功能的增强，不利于芯片达到最佳性能。目前，这已经成为限制处理器设计发展的主要因素。

新应用的出现需要新的软件解决方案。与 CISC 发展历程相一致，新指令自然地满足了新应用的需求。例如，计算机图像处理器就已受处理器速度所限，处理器性能也受到指令执行率的影响，而新指令可以经过优化而为提高性能服务。

#### 3.7.2 指令集扩展

已有的指令集现在仍然保留着，另外又增加了一些新指令，如果处理器能够兼容指令集，那么新应用就可以充分利用附加指令的优势。处理器向上需要能够兼容新应用，向下需要兼容已有的代码。

扩展指令集是防止软件过时的一个重要方法。处理器设计并不能完全达到新产品的最佳性能，然而，性能提升其实需要一个折中的方案，该方案需要处理器在维持已有应用的基础上也能适应新应用的要求。

一款新处理器要投入市场需要较高的成本，对处理器设计师和终端用户方面的总投资需要合理评估。处理器设计的新技术相应地带来了处理器性能和功能的增强，这些都使得现在的处理器比从前的性价比更高。但是如果利润没有，那么再好的处理器也不可能上市。

由于 Intel 808x 系列的处理器已经主导了 PC 市场，所以其他处理器的设计很难推向市场，PC 抑制了存储器、软件和处理器技术创新方面的平衡。就是说，通过整合各种技术，处理器可以获得更佳的性能。

### 3.8 RISC 指令集体系结构

总之，RISC 设计注重简单，这里的简单不仅仅指的是指令集，也包括上述三种主要技术的简化。处理器逻辑为获得更快的速度而简化，存储器划分为程序存储器和数据存储器以加快访

问速度，编译产生的软件也整合到了基础设计之中。

### 3.8.1 微代码

我们已经探讨过程序的三种主要技术，软件程序倾向于执行某个小区域的代码，此执行属性独立于总体的程序结构。这是程序的总体特点。

另一个主要方面就是有一小部分指令会在程序执行期间重复使用，像数据移动、布尔运算和算术指令这些基本指令在所有指令中是最主要的。同样，这个趋势也是独立于软件的，执行指令集会针对不同的应用而改变，但是它只是全部指令集的一个子集。

局部性的第三个层次是在微代码级。正如我们前面所讨论的，CISC 处理器以微代码指令序列的方式执行处理器指令，在所有的微代码指令中，同样也有一个较小的指令子集反复使用。

### 3.8.2 微指令周期

微代码的概念是由 IBM 公司于 20 世纪 60 年代在 System/360 系列计算机中提出的。微码是一个小的软件程序，存储在处理器的存储器中。微代码指令已经整合到基本的处理器逻辑功能之中，机器指令则被转换成一系列微代码指令。

微指令控制着处理器的内部逻辑以及指令执行，就好像是其自身程序一样。机器指令周期定义为执行时所需要的微指令数目。例如简单的加法指令（ADD）只需极少的微指令；然而，乘法指令（MUL）执行时可能需要 32 个周期。

MUL A, B	LOAD A
	LOAD B
	SHIFT A
	SHIFT A
(重复八次)	

相应地，执行更复杂的指令时需要更多的微码周期。含有间接寻址或者相对寻址模式的指令执行就是典型例子。这些类型的指令只在处理程序的某个子集时有用。对于任何已知的程序，只能应用到指令集的一部分指令。

### 3.8.3 专用指令

专用应用要获得最佳性能，其包含的专用指令必须能够以最快的速度执行。如果采用复杂的处理器逻辑和微码指令，单纯通过硬件技术、软件技术和存储器技术的改进来达到理论最佳性能是十分困难的。

同样，阿姆达尔定律和摩尔定律中包含了相应的解决方案。依阿姆达尔所言，性能与提高性能的相关技术已采用的时间成比例增长；而摩尔定律指出微处理器逻辑将变得越来越快，越来越复杂。

多数汇编指令执行时仅仅需要微码指令的一个子集，通过减少微码周期数，总体性能会有所提高。

### 3.8.4 单周期指令

RISC 指令集体系结构的一个主要创新就是提出了单指令周期的概念，该概念是基于阿姆达尔定律的基本原则的。RISC 仅仅包含典型 CISC 处理器中常用的微码指令。在这个意义上说，RISC 处理器中并不包含微码，RISC 指令集本身就可以说是微码指令集。

依据局部性原理，RISC 处理器关注于执行 CISC 处理器的“微码”。除了分支指令，其他所有的 RISC 指令都在单周期中执行。这样就将如下等式中的第二项减少到可以近似为 1（采用超标量设计可以在一个周期内执行多条指令。）：

CISC Time - pgm = Inst - pgm * cycles - inst * time - cyc
RISC Time - pgm = Inst - pgm * 1 * time - cyc = Inst - pgm * time - cyc
注：不是所有的指令都只执行 1 个周期

采用优化的 C 编译器，相比 CISC 可以大大减小第一项。如果采用简化逻辑，第三项也可减小。在同样的半导体技术下，RISC 相较于 CISC 可以显著提高性能。

上述等式对于 CISC 和 RISC 都适用。CISC 设计主要致力于减小第一项，这就减少了程序中被执行指令的数量，同时也会对存储器和编程器价格昂贵的问题产生影响，这两个经济指标也会随着相关技术的进步发生相应的改变。然而，它们在 CISC 体系结构的发展史上扮演着极为重要的角色。

### 3.9 处理器逻辑

处理器由多个功能模块构成，每个单元都有一个信号传播延迟时间，这个时间就可以定义为功能模块的门延迟时间之和。图 3-26 给出了一个多层逻辑的示例以及相应的延时。

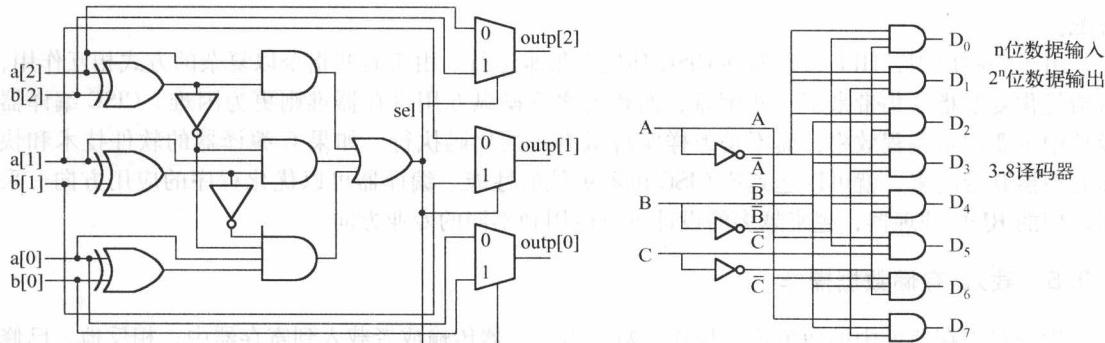


图 3-26 逻辑门延时减少

半导体工艺技术的进步降低了晶体管的开关时间，更快的开关时间减少了门延迟时间的同时又增加了时钟频率。以上这些是对于同步处理器而言的，不论该处理器是基于 RISC 的还是基于 CISC 的。

RISC 致力于在处理器逻辑层提高性能。通过减少指令的数量，处理器逻辑可以通过更少的逻辑门来实现。逻辑门数量的减少同时又降低了信号传播延时时间，反过来，这又允许有更高的时钟频率。

#### 3.9.1 同步逻辑

在同步处理器设计中，每一个模块的时钟都由机器周期率决定。一条指令的执行周期就是执行中各个模块耗费的时钟周期的总和。指令执行率可以通过两种方式来提高，即降低功能模块中的门延时或者减少总体时钟周期数。

在最简单的形式中，如果处理器逻辑在一个模块中完成，那么就可以在一个时钟内完成一个周期。在 RISC 中，这通过降低逻辑的复杂性来实现。RISC 的指令集经过特殊定义以使所有的指令都可以在单时钟周期中执行完毕。

### 3.9.2 寄存器堆

RISC 体系结构虽然理论上说有非常高的性能，但是在 20 世纪 50 年代这却难以实现。当时的处理器不仅速度慢，而且价格高，而 RISC 需要较大的存储空间。RISC 理论的一个重要方面就是需要十分有效的高层语言（HLL）技术，而当时的技术还不能开发出必要的优化编译器。

### 3.9.3 正交寄存器

RISC 有许多非常优秀的特点，例如指令在单周期内执行（分支指令除外），这在寻求最佳性能的过程中大大简化了代码的生成过程。另一方面，寄存器堆是通用的，它们是可替换的，或者说是正交的，如图 3-27 所示。

正交寄存器意味着寄存器在使用过程中是可替代的。编译器并不在意使用的是否专用寄存器，可以更有效地使用任何寄存器来存储中间数据值。

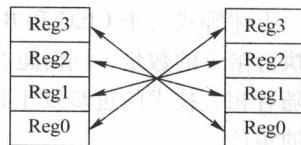


图 3-27 正交寄存器组映射

### 3.9.4 寄存器优化

CISC 编译器将中间数值存储在临时存储区域中，增加了存储器操作的步骤，同样也增加了程序执行时间。阿姆达尔定律指出，指令越少，意味着性能越高。RISC 编译器可以优化寄存器分配。

由于一些原因，HLL 编译器在 CISC 环境下很难运行。由于有些指令以复杂的方式相互作用，这给优化复杂指令集带来了一些困难，而相比之下操纵专用寄存器堆则更为困难。CISC 编译器设计中主要注重编程效率，而不是怎样实现最有效的代码执行。如果 C 编译器的软件技术和快速处理器联合起来，就可以达到比 CISC 机器更佳的性能。编译器可以优化程序的应用方向。采用通用的 RISC 处理器，越来越多的设计可以应用到不同的专业方向。

### 3.9.5 载入/存储数据操作

指令对寄存器堆中的数值进行操作，数据由存储器传输或者载入到寄存器中；相反地，已修改的寄存器数据也会保存到存储器中，如图 3-28 所示。这是 RISC 技术的一个重要特点。由于寄存器中存有数据，这样就可以让指令在一个周期内执行完毕。

正交寄存器同样也会影响指令执行率。指令不需要专用寄存器以一定的顺序加载，载入存储器或者从存储器中取出数据都无需等待专用寄存器处在可用状态。



图 3-28 RISC 数据寄存器

一旦寄存器载入了执行运行的数据，其中间结果就可以写入任何一个可用的寄存器，后续的指令可以对这些数值进行操作。就指令序列可以完成的程度而言，如果操作从寄存器中获得数据，那么性能就可以达到指令周期时间的最大值。

## 3.10 处理器功能划分

正如我们所了解的，在任何时候进行处理器设计，都是在寻求可行技术的平衡，以获得最佳性能。以后的处理器（如 CISC）是多种功能单元的结合。处理器的功能划分设计允许工程师在当时的技术条件制约下打造一款更加复杂的处理器，这将比采用单一的独立逻辑更具实际意义。一个折中的办法就是牺牲一小部分的潜在性能换取设计能力上的巨大提升。微控制器设计中，在可获得的最佳性能和低成本之间寻求一个折中的方案是常见的平衡办法。

### 3.10.1 指令流水线

功能单元由控制逻辑控制，它们依据时钟信号进行同步，就像指令执行的装配线一样。指令执行周期的每个阶段都有部分结果被抽取出来，就像水流过管道一样。指令周期的时间就是从指令进入流水线到离开流水线的总时钟数。

图 3-29 所示为一个按固定速率流过水流的管道。如果管道的直径扩大，那么在同样的压力下就可以通过更多的水流。而如果压力增加，水流的速度也会加快；如果水压固定，那么扩大管道的直径就可以让更多的水流通过。

### 3.10.2 执行单元

指令流水线是 RISC 体系结构的一个重要特征。RISC 结构将程序和数据分别存入独立的存储器空间中，这有着重大的含义。在程序存储空间中，指令被顺序储存，如图 3-30 所示。

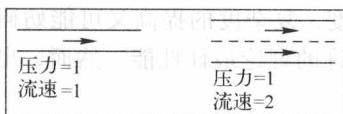


图 3-29 指令流水线模拟

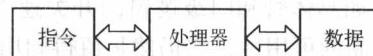


图 3-30 RISC 存储器体系

CISC 体系结构则不同于此，它的指令和数据都在同一个存储空间中。当然，其指令长度不尽相同，如图 3-31 所示。这意味着指令并不是有序地排列在一个字边界上，只有当第 n 个指令被取出来解码后，下一个指令 n+1 才有可能计算出来。

不同层级的存储器速度不同，这主要受技术因素的影响。然而，为了方便从存储器中提取出指令，RISC 体系结构增加了一个重要的优势。通常，在当前指令执行结束之前就可以得知下一个指令的地址，这适用于除分支类型指令之外的指令执行。

### 3.10.3 流水线级

指令一级一级地得到执行，只有一个功能单元在及时处理问题。如果从一个单独的处理器逻辑单元发展到由两级组成的处理器逻辑单元，可以成倍地加快指令执行的速率，如图 3-32 所示。但是这在现实中并不实际，因为在分支指令中会产生处理器周期增量。

对处理器进行分区设计自然地要追求各功能模块的充分利用。如果执行单元能够一直保持忙碌，那么效率是可观的。RISC 体系结构自然地达到了这个目标：最佳的性能和最高的效率。

性能的急剧增加使流水线技术变得更富有吸引力。如果一个功能块的延时是 20 ns，那么连续的指令执行可能会花费 40 ns。但如果两个功能块可同时保持运行，那么处理器可以在 20 ns 内完成一个复合的指令执行，如图 3-33 所示。

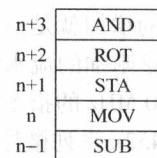


图 3-31 存储器中的指令顺序存储

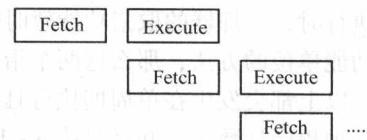


图 3-32 两级指令流水线

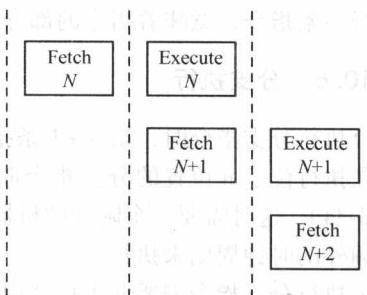


图 3-33 有效指令执行周期

处理器逻辑会花费 40 ns 来处理这个指令，然而，指令的每一部分都处在执行阶段，一部分在取指阶段，一部分正在执行。取指和执行代表了完整的指令周期。这样，指令周期就基本上等同于机器时钟周期了，因为指令执行最多需要两个机器时钟周期来执行。RISC 指令主要由单周期指令操作组成。

增加流水线的流水级数可以有效提高性能。要达到这个目标，合理安排各技术要素显得更为复杂，不仅在硬件方面，同时也要考虑到软件和存储器方面。另外，这三项主要技术仍然要寻求一个平衡点，以在约束范围内获得最大的性能。

在微控制器的设计中，RISC 指令集体系结构限制在七级流水的范围内。简单的微控制器可能只涉及取指和执行两级流水，而高性能的处理器则可能会将执行级拆分为更多的流水级。

利用流水线技术的关键在于以同等的半导体技术追求性能的增长。如果指令集可以在 20 ns 内有效执行，那么性能就会加倍。事实上，这种倍增效应（对于一个两级流水线）只是理论上的最大值。实际的程序包含分支指令，而分支指令需要额外的周期来执行。但是，如果能得到 80% 的性能增加，那么设计复杂度的缓和增长也是值得的。

功能单元采用新逻辑可能增加处理器设计的复杂度，复杂度的提高又可能妨碍产品达到半导体技术（比如晶体管和门级逻辑的开关速率）所允许的理论最佳性能。然而，出于设计能力的价值以及产品的可用性，它仍然是值得利用的。

### 3.10.4 流水线吞吐量

如果固定时钟频率的处理器性能可以通过最小的努力提高为原来的 200%，那么其所带来的设计复杂度的增加就是值得的。许多流水线级是与 RISC 中最小化逻辑和单周期指令的概念背道而驰的，但是附加的流水级可以带来更佳的性能。例如对于 100 MHz 的时钟频率来说，5 级流水线能提供 400 MHz 的指令吞吐量以及 80% 的流水线效率。

PIC18F452x 系列微控制器采用了简单的两级流水线，使得最佳性能有效地翻了一番。它是取指和执行阶段的一个简单重复，取指以及连续指令执行的交叠使得其执行率接近每时钟周期执行一条指令。但是由于分支型指令的存在，其性能不可能如理论上所描述的那样翻一番。

### 3.10.5 顺序执行

在一个线性处理的处理器中，整个指令周期中只有一个逻辑功能单元处在活动状态。在指令执行时，处理器的取指功能暂时停止；而在取指时，指令则暂不执行。如果能找到同时利用两个功能单位的方法，那么这两个指令就可以同时部分执行。

以上都会发生在单周期执行且顺序存储的指令上。处理器逻辑本身在执行第 n 条指令时会自动增加程序计数器，并且对第 n+1 条指令执行取指操作。这个过程不受程序员的控制。

这些带来了每周期执行一条有效指令的实际效果，是流水线设计中最重要的理念。每周期执行一条指令，意味着指令的部分执行，代表了单指令周期。

### 3.10.6 分支执行

执行分支指令时，第 n+1 条指令并不就是真正的下一个指令（紧随 n 的指令）。因为处理器在执行位于 n 位置的分支指令时，已经取出了第 n+1 条指令，该指令即为错误的指令（见图 3-34）。这时需要一个附加的机器周期来取出 n+4 的分支地址。一旦新指令被取出，就需要一个额外的时钟周期来执行。

执行分支指令需要两个指令周期，因为直到 n+2 处的取指完成之后，新地址才可能计算出来。分支指令越少，用户越有可能获得最佳性能。因此，如果把性能考虑在内，基于 PIC 微控制

器进行设计的一个主要目标就是减少分支指令的数量。

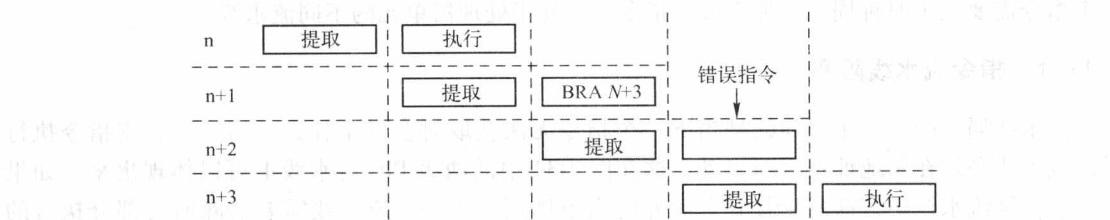


图 3-34 分支指令执行延时

### 3.11 五级流水线

五级流水线常见于多级流水线 RISC 体系结构中。此数量的流水级既能提供显著的性能提升，又可将处理器逻辑的额外开销限制在一个可以忍受的范围内。图 3-35 所示为典型的五级流水线。

微控制器的成本从根本上来讲是和工艺成本紧密相连的。MicroChip PIC 的内核逻辑相对简单，其晶片尺寸非常小，因而直接导致了其较低的生产成本。在 8 位商用微控制器设计中，PIC 获得了极大成功：简单，精美，低成本。

MIPS 4Kc 采用了取指、执行、存储器操作、排列和写回五级流水。如果处理器逻辑的时钟频率为 2 GHz；如果同 RISC 指令集体系结构的概念联系起来，那么 MIPS 4Kc 将获得相当高的性能。图 3-36 所示为 MIPS 4Kc 的流水线示意图。

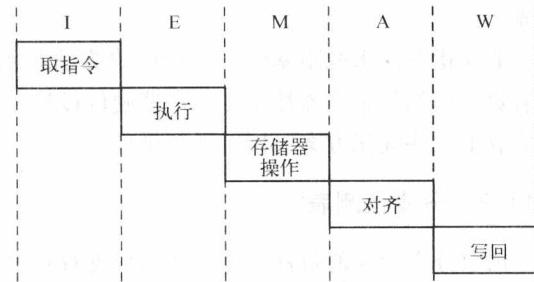


图 3-35 五级流水线

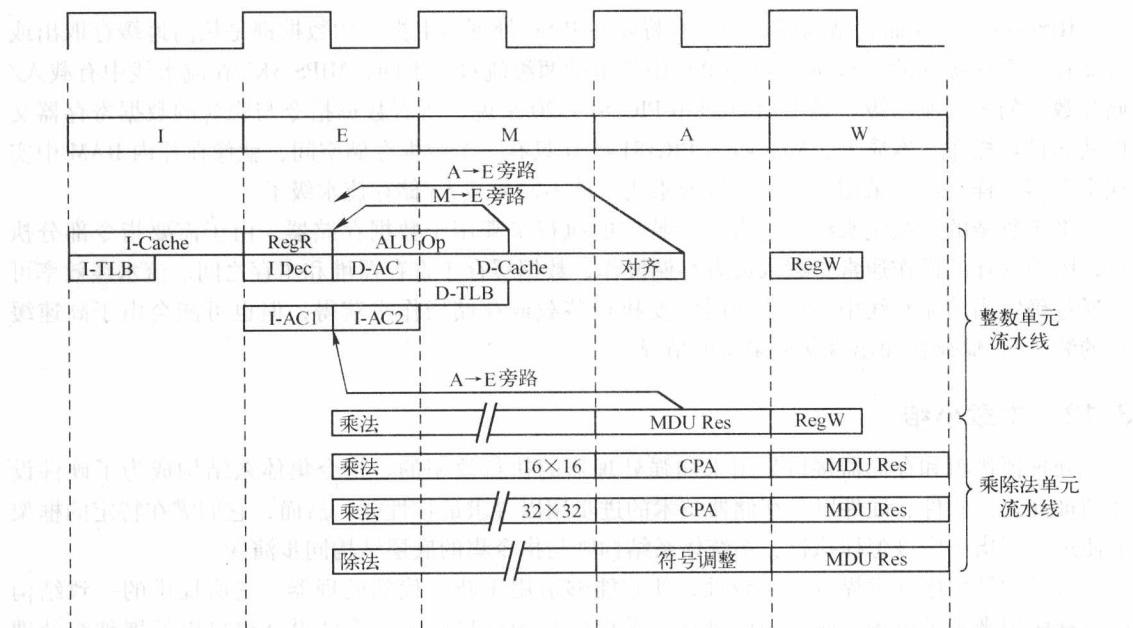


图 3-36 MIPS 4Kc 内核流水线（经 MIPS 科技股份有限公司许可使用）

指令重叠原理相同，对于 MIPS 4Kc 来说，意味着指令周期功能部件的重复运行。如果执行一个指令需要五个时钟周期，那么五个指令需要处于处理器单元的不同流水级。

### 3.11.1 指令流水线阻塞

流水线遇到的一个主要问题是当下一个指令无法获取时会发生什么。当一个分支指令执行时，流水线必须在新地址上重新启动。这在图 3-34 中的两级 PIC 流水线上可以体现出来，如果在一个五级流水线中，这意味着五个额外的指令周期会丢失，流水线需要清除所有部分执行的指令，并在新地址上重启。这极大地影响了性能。

如果分支指令花费了 5% 的时间，那么它会使潜在的性能增长降低 20%（见图 3-37）。考虑到分支指令会使多级流水线的性能大打折扣，RISC 中采取多种措施减少分支指令。这些措施既可以在硬件设计中完成，也可在软件编码中完成。

$$\begin{aligned} 100_{\text{inst}} * 1_{\text{inst/cycle}} &= 100_{\text{inst-cycles}} \\ (95_{\text{inst}} * 1_{\text{inst/cycle}}) + (5_{\text{inst}} * 5_{\text{inst/cycle}}) &= 120_{\text{inst-cycles}} \end{aligned}$$

图 3-37 流水线分支开销

消除指令流水线阻塞最简单的解决方法是消除分支指令。通过优化软件，许多指令分支可以清除。这就是编译器技术和处理器硬件设计相互作用的地方。RISC 处理器中的 C 和 C++ 编译器采取了一些优化方案，用以展开循环。

### 3.11.2 分支预测表

预取分支指令的目标地址，使用分支查找表可以实现。当遇到分支时，分支地址指令已经被预取。如果分支被采取，那么该指令已经取出，并且可以直接送往执行单元。这样就不会错失时钟周期，而流水线也会一直保持饱和状态。这会增加目标对象的文件大小，但是它同时也提高了处理器执行的效率。这样一种平衡并不总是固定的，而会因应用程序的不同而不同。

### 3.11.3 数据流水线阻塞

RISC 以载入和储存结构闻名。所有将要在 RISC 处理器上操作的数据都是从高速缓存取出或者储存到高速缓存的。与 Microchip PIC 中简单的两级流水线不同，MIPS 5Kc 在流水线中有载入/储存数据的专用流水级。对于 Microchip PIC18F4520 来说，所有数据指令与隐含的数据寄存器文件共同进行操作。本质上，Microchip PIC18F4520 只有一个数据存储空间，就像在片内 RAM 中实现寄存器一样。如果采用简单的两级流水线，就不需要载入/储存流水级了。

至于复杂的多级流水线，会有一个特定的执行步骤用于数据存储器。由于需要指令部分执行，因而这样的操作通常是载入或者存储操作，数据缓存于寄存器堆和主存之间。流水线效率可以通过确定当前流水线中的指令何时需要执行装载或存储操作来获得，但也可能会由于高速缓存的装载/存储操作而出现保存阻塞的情况。

## 3.12 本章小结

处理器逻辑和存储器接口是用来加强处理器的执行效率的。指令集体体系结构成为了硬件设计的推动力，软件、硬件以及存储器技术的进步用以寻求最佳性能。然而，它们需在特定的框架下使用，即指令应该怎样执行。系统体系结构则与指令集的底层结构同步演变。

这为软件程序开发提供了一致性，使它能够适用于新一代的处理器。它所提供的一致结构以各种应用来满足市场，而市场凭借这些关键的结构凝聚起来，软件设计也可以发展到跨处理器平台。只要保持指令集体体系结构的完整性，科技进步就可以不断地推动性能的提高。

同时，当技术的聚合可以为应用带来更好的优势时（对市场而言），指令集体体系结构的分支也会出现。这可通过采用附加的专用指令或者改变指令集体体系结构本身来实现。

改变指令集体体系结构可以大大提高其性能，因为在解决方案中的三个关键技术全部可以得到最充分的利用。处理器设计工程师就可以从一些约束条件中解脱出来，这些约束条件使得设计者维持所有向下兼容，同时成为限制处理器设计创新的主要因素。

## 习题

1. 请说明摩尔定律的具体内容。
2. 在发展新的半导体技术中，最为关键的产品是什么？
3. 什么是使能技术？
4. 解释阿姆达尔定律。
5. 在系统设计中设计抽象是如何应用的？
6. 请列举 RISC 和 CISC 之间的主要区别。
7. 说出三种减少程序执行时间的方法。
8. 微码的定义是什么？
9. 什么是程序的局部性原理？
10. 列举基于 RISC 设计的各级存储器。
11. 高速缓存的用途是什么？
12. 汇编和编译器代码之间的主要区别是什么？
13. 若缓存的高速访问时间是 500 ps，主存访问时间为 2 ns，文件系统访问时间为 8 ns，那么存储器延迟是多少？
14. 列举两个指令集扩展的例子。
15. 描述微码以及它的用法。
16. C 优化编译器的用途是什么？
17. 拥有正交的寄存器组意味着什么？
18. 为什么指令流水线能够提高性能？
19. 分支指令的执行对流水线会产生什么影响？

## 参考文献

- Intel Web Site: <http://www.intel.com/design/mcs51/manuals/272383.htm>  
 Tensilica Web Site: [http://www.tensilica.com/products/xtensa\\_LX.htm](http://www.tensilica.com/products/xtensa_LX.htm)  
 MicroChip Data Books: <http://www.microchip.com>  
 MIPS Web Site: <http://mips.com/products/overview/>

处理器	位数	寻址能力	时钟频率	功耗
Z80	8	64 KB	4 MHz	10 mW
8086	16	1 MB	4 MHz	100 mW
386	32	4 GB	10 MHz	1 W
奔腾	32	4 GB	100 MHz	10 W

## 第4章 |

Embedded Microcontrollers and Processor Design

# 微控制器功能

- 本章目标：了解微控制器功能

- 主要内容：

1. COMS 工艺介绍
2. 微控制器中的不同类型存储器介绍
3. 微控制器的基本硬件特征介绍
4. 常用微控制器外设功能介绍

## 4.0 设备功能

通用单片机都具有一套从典型应用需求发展而来的最基本的设备功能，像 I<sup>2</sup>C 和 UART 这样的工业标准功能就是典型的例子。微控制器也可以包括面向专门应用的功能，通过二者有机的结合，使得控制器应用专注于基于标准接口进行开发。

纳入芯片的功能数量取决于该芯片所面向的应用。低成本设备拥有较少的功能，而越是昂贵的设备越是集成了更加强大的功能集合。市场价格和应用所需要的功能共同决定了微控制器的设计方向，图 4-1 展示了在价格和设计所包含的功能之间所具有的最优区域：

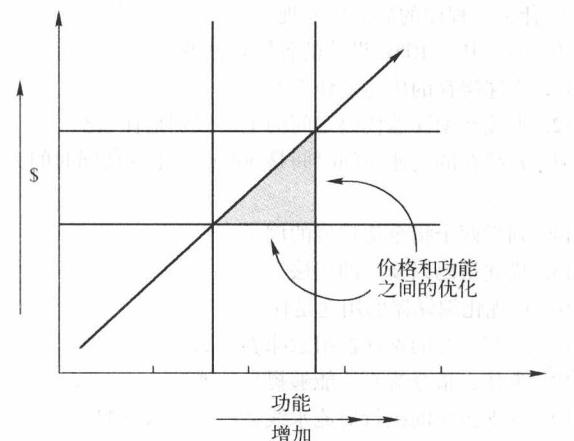


图 4-1 微控制器市场的设计约束

## 4.1 晶体管工艺

双极晶体管工艺比互补金属氧化物半导体 (Complementary Metal-Oxide Semiconductor, CMOS) 工艺出现得早，它被用于标准的小规模集成电路和中等规模集成电路逻辑器件。高速器件使用肖特基晶体管进行设计，而最小功耗设计则需采用低功耗晶体管进行设计；两者结合可以用于设计比标准器件功耗低、速度快的器件。图 4-2 列出了标准 2 输入与非逻辑功能的几款设计。

### 4.1.1 CMOS 晶体管

目前已经开发出多种 CMOS 设计工艺。嵌入式微控制器根据特定的应用领域，采用不同的工艺。对于电池供电的微控制器设计，低功耗是主要的设计准则；而对于高性能的设计，速度则变成了设计的目标。图 4-3 列出了主要的 CMOS 工艺及其典型应用。

标准	低功耗	高速	低功耗及高速
7400	7400L	7400S	7400LS

图 4-2 SSI 与非逻辑

工艺	应用
CMOS	标准
CHMOS	低功耗
Bi-CMOS	混合信号
XCMOS	闪存

图 4-3 CMOS 工艺类型

CMOS 晶体管的工作电流比双极型晶体管小，这是由 CMOS 晶体管的电容特性造成的。图 4-4 所示为一个 CMOS 晶体管原理图，它包括一个栅极、一个源极和一个漏极。 $V_{dd}$  取代了  $V_{ee}$ ， $V_{ss}$  为电源地。

图 4-5 显示了一个由一对互补 MOS 晶体管（CMOS）组成的反相器。在静态情况下，这两个晶体管都是“关闭”的，也就意味着基本上没有电流；当输入电压  $V_{in}$  转换的时候，电压变化的瞬间会有电流流过。



图 4-4 CMOS 原理图

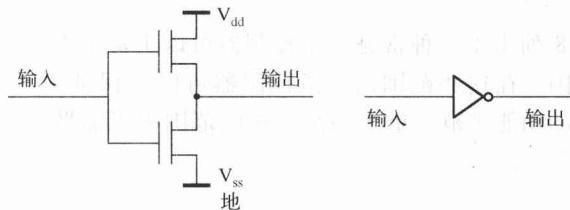


图 4-5 CMOS 反相器原理图

#### 4.1.2 CMOS 功耗

对于包含大量晶体管的微控制器，低功耗设计是很有必要的。在双极晶体管工艺中，总有一部分晶体管处在开启状态（逻辑 1），这就意味着会有电流流过。但对于 CMOS 却不是这样，从实质上来说，CMOS 除了逻辑状态转换以外，其他时刻都没有电流流动。

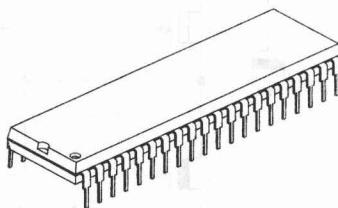
CMOS 器件的功耗与开关频率成正比，这一点可以在个人电脑设计上得到验证。微处理器的时钟频率越高，对散热的要求就越高。对于图形显示微控制器，由于器件数量的规模大，要求的性能较高，因此需要散热片。依据复杂性的不同，主板上的系统芯片往往需要某种形式的散热装置。

在嵌入式微控制器的设计中，功耗是限制系统性能的一个主要因素。在基于微控制器的电路板设计中，依据系统设计需求，通常增加诸如散热片等任何形式的机械设备都是不允许的，特别是当微控制器的板卡在物理上受限制时，应该去掉多余的机械部分。

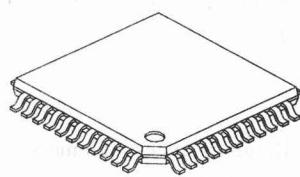
#### 4.1.3 封装

最常见的微控制器封装材料是塑料，它也成了事实上的工业标准。为了降低封装成本，封装的引脚数量也是标准化的，如果为工程定制一个专门的封装形式将会带来很高的费用。

商业微控制器用于大批量生产，塑料便是专门为此目的而设计的低成本材料。封装的引脚数量取决于应用的要求，I/O 功能越多，需要的引脚数量就越多，如图 4-6 所示。引脚数量最少的塑料封装是成本最低的封装形式。



a) 40 引脚 PDIP 封装



b) 44 引脚 TQFP 封装



图 4-6 (经微芯科技股份有限公司许可使用)

基于 SoC 的微控制器可以用塑料封装，也可以用陶瓷封装，如图 4-7 所示。对于大批量的商业应用，也要考虑相关决策。多数大批量 SoC 设计采用方形扁平封装方式，允许在晶粒四周进行晶粒粘接 (die bonding)，这对保持芯片电压以及平衡 I/O 端口电流具有明显的作用。

#### 4.1.4 工作温度范围

图 4-8 列出了三种常见的微控制器可以正常工作的温度范围。在这些范围内，微控制器可以“保证”实现发布的功能规范。重要的是，在此范围内无需散热器也能工作。

### 4.2 存储器工艺

图 4-9 列出了四种主要的存储器工艺，即读写存储器 (Read-write Memory, RWM)、非易失性读写存储器 (Non-volatile Read-write Memory, NVRWM 和 FLASH) 和只读存储器 (Read-only Memory, ROM)。RWM 和 NVRWM 实现的电路设计工艺不同，需要依据应用来选择。ROM 一般基于掩膜编程制造工艺。

封装类型	
PDIP	塑料双列直插封装
SOIC	双侧引脚小外形封装
TSOP	薄形小尺寸封装
PQFP	塑料方形扁平封装
PLCC	塑料式引线芯片承载封装
TQFP	薄形四方扁平封装

图 4-7 封装类型

温度范围 (°C)	设计用途
0°C ~ +70°C	商用
-45°C ~ +85°C	工业
-55°C ~ +125°C	军事

图 4-8 标准温度范围

读写存储器 (RWM)	读写存储器 (RWM)	非易失性读写存储器 (NVRWM)	只读存储器 (ROM)
随机存储器	非随机存储器	EPROM	掩膜编程
SRAM	FIFO	E <sup>2</sup> PROM	
DRAM	LIFO	Flash	
	移位寄存器		
	CAM		

图 4-9 存储器类型

#### 4.2.1 DRAM

如同它的名字一样，DRAM (Dynamic Random-Access Memory, 动态随机存储器) 如果没有及时的刷新，将会丢失存储信息。随着时间的推移，保存逻辑 1 的位可能读出来的是逻辑 0，这就意味着 DRAM 需要不断刷新来保持逻辑 1 的状态。DRAM 的单元只有一个晶体管，往往垂直排列。这种先进的工艺可以极大地压缩几何空间，因此便有了高密度 DRAM 的说法。

在微处理器的设计中，DRAM 只用于专门的特定应用。虽然是高密度，但它的基本制造技术是基于模拟电路的。将 DRAM 和数字微控制逻辑进行整合需要非常复杂的制造工艺。在实际的应用中，DRAM 和微控制器独立使用，并非集成在同一芯片上的。

#### 4.2.2 SRAM

SRAM (Static Random-Access Memory, 静态随机存储器) 单元不刷新也能保持它的值。从本质上讲，这是一个触发器的形式。典型的静态 RAM 由 6 个晶体管单元组成，如图 4-10。

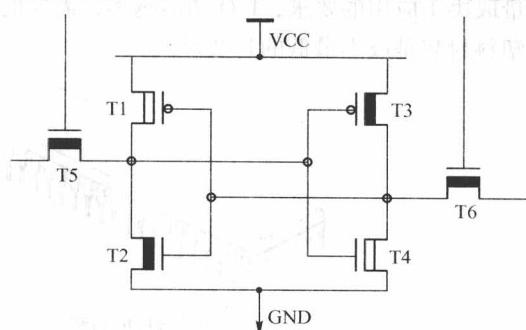


图 4-10 标准 6 管 CMOS SRAM 单元

这使得一个相同容量的 SRAM 比 DRAM 大很多。

基于 RISC 的 PIC18F4520 单片机集成了 1536 字节的 SRAM。例如，基本 i8051 单片机具有 256 字节的片上存储器。对于面向高性能应用的嵌入式 SoC 设计来说，16K 字节的 SRAM 并不少见。按照摩尔定律，新设计的处理器可以集成更大容量的 SRAM 存储器。

SRAM 主要有四种类型，每种类型都有特殊用途。后进先出存储器（Last In First Out，LIFO）用于地址堆栈，图 4-11 给出了 LIFO 在一个 8 位 PIC 微控制器中的应用，指令地址被“压入”（pushed）堆栈，或从堆栈“弹出”（popped）。先入先出（First In First Out，FIFO）随机存储器用于实现程序指令缓冲。再者，移位寄存器主要用于对处理器逻辑功能模块中的数据实现序列化。这些不同基本工艺的实现仍然基于 6 个晶体管的 SRAM 单元。

#### 4.2.3 NVRWM

当应用程序所需的数据信息需要加电固化时，一般采用 EEPROM 存储器。电视机的配置数据就是一个例子，接收器的选项由观看者进行设置，如频道的选择等，当电视电源插座上电时，之前选择的频道依然存在。

微处理器一般包含两种类型的 NVRWM，一种是 FLASH 存储器，主要用于储存程序；另一种是 EEPROM，主要用于非易失性数据存储器。FLASH 有时也称为闪存，能够将数据按块组织为存储器字，快速按存储器字进行编程。例如 Micro 的 PIC18F4520 等新型的处理器芯片，支持通过指令集来进行 FLASH 编程，方便用于远程程序更新。

除  $t_{acc}$  参数外，还有三个主要参数与 FLASH 和 EEPROM 有关，其中两个参数与时间有关，是编程或擦除存储器字所花费的时间；第三个参数是擦除或重写能够执行的总次数。图 4-12 给出了 PIC18F4520 微控制器的 EEPROM 和 FLASH 工艺参数。

	参数	最小值	典型值
EEPROM 数据存储器	字节的耐擦写能力 在刷新之前的总擦/写周期数	100K 1M	1M 10M
闪存程序存储器	单元的耐擦写能力	10K	100K

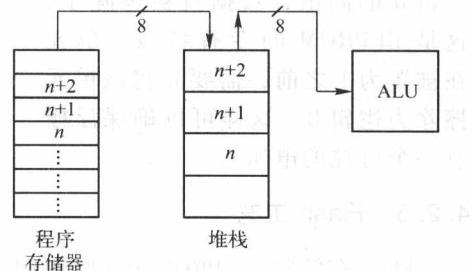
图 4-12 PIC18F4520 E/W EEPROM/FLASH 规范

时间就是金钱，在产品的设计过程中，编程时间是尤为重要的。软件必须要下载到微控制器进行调试，代码下载程序越快，工程师就可以更快地继续工作。

第二个重要的方面是编程/擦除周期的次数。EEPROM 工艺限制了单元电路可以被编程的次数，它总是先擦除再重写，因此也称为编程/擦除周期。EEPROM 支持 100 000 次的编程/擦除周期，这个数字是相当大的，假设一款产品的生命周期为 10 年，数据每小时更新一次，其共需 43 810 个更新周期。

#### 4.2.4 EEPROM

EEPROM 单元是基于浮动栅原理的（见图 4-13）。在编程过程中，浮栅上的充电电压增加到逻辑 1 的阀值，这是通过将编程电压提高到 5 V 以上来实现的。电压的升高会导致电荷从薄隧道



氧化层向浮栅进行迁移。当芯片电源关闭后，栅极电压仍然存在。

在擦除周期，充电电压降低至逻辑0的阈值，数据就被擦除了，这是EEPROM的专有特点。单元在被置为1之前，需要先将该单元擦除为逻辑0，这样可以确保浮栅有一个可控的电压。

#### 4.2.5 Flash 工艺

Flash存储器是EPROM和EEPROM工艺的结合，并具有快速编程的能力，可以在一个周期内将连续地址上的大量存储单元进行编程。例如，PIC18F4520一次擦除/重写的块大小为64字节。块的大小越大，FLASH编程就越快。

#### 4.2.6 ROM

只读存储器基于单晶体管单元设计，是存储程序使用最多的存储器。和DRAM的主要区别在于，存储位的逻辑1或0的状态能够永久保持，其数据存储状态是在生产的时候创建的，ROM也被认为是掩膜可编程的。

### 4.3 硬件特性

商业微控制器都尽可能设计成为通用微控制器。面向给定的应用，微控制器能够提供专门的配置，这些能力的扩展使得它能够扩展潜在的市场。例如，针对低成本应用的微控制器设计，可能使用一个简单的晶体振荡器，或者使用内部生成的RC时钟。

#### 4.3.1 配置字

许多特定的配置选项都是由硬件选择的。在上电的时候，这些特定的选项决定了该设备的特性，图4-14所示为PIC18F4520配置字设置。在对微控制器的软件编程时，这些硬件功能被启用（或停止）。这些功能选项是故意固定在硬件中的，以使软件无法改变它们。例如，我们不希望无意中禁用芯片的振荡器。

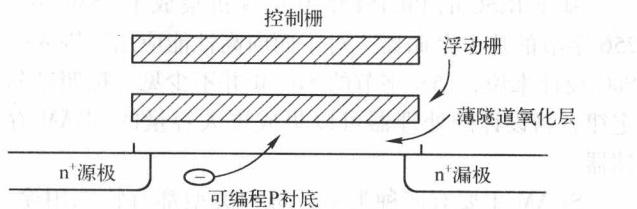


图4-13 EEPROM 浮动栅

文件名	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	默认/非编程值
300001h CONFIG1H	IESO	FCMEN	—	—	FOSC3	FOSC2	FOSC1	FOSC0	00-- 0111
300002h CONFIG2L	—	—	—	BORV1	BORV0	BORV1	BORV0	PWRREN	---1 1111
300003h CONFIG2H	—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN	---1 1111
300005h CONFIG3H	MCLRE	—	—	—	—	LPT1OSC	PBADEN	CCP2MX	1--- -011
300006h CONFIG4L	DEBUG	XINST	—	—	—	LVP	—	STVREN	10-- -1-1
300008h CONFIG5L	—	—	—	—	CP3	CP2	CP1	CP0	---- 1111
300009h CONFIG5H	CPD	CPB	—	—	—	—	—	—	11-- ----
30000Ah CONFIG6L	—	—	—	—	WRT3	WRT2	WRT1	WRT0	---- 1111
30000Bh CONFIG6H	WRTD	WRTB	WRTC	—	—	—	—	—	111- ----
30000Ch CONFIG7L	—	—	—	—	EBTR3	EBTR2	EBTR1	EBTR0	---- 1111
30000Dh CONFIG7H	—	EBTRB	—	—	—	—	—	—	-1-- ----
3FFFFEh DEVID1 <sup>(1)</sup>	DEV2	DEV1	DEV0	DEV4	REV3	REV3	REV1	REV0	xxxx xxxx
3FFFFFh DEVID2 <sup>(1)</sup>	DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3	0000 1100

图4-14 PIC18F4520 配置字（经微芯科技股份有限公司许可使用）

### 4.3.2 振荡器类型

工程师通常可以选择几种不同类型的外部时钟发生器，这主要依据应用程序进行选择。图 4-15 中的图表列出了 Microchip PIC18F4520 系列可以使用的振荡器类型。一般来说，要求的计时越精确，外部元件的成本就越高。

设计	类型
LP	低功耗振荡
XT	标准晶体振荡
HS	高速晶体振荡
HSPLL	具有PLL使能的高速晶体振荡
RC	RA6中的外部阻容振荡，具有FOSC/4频率输出
RCIO	RA6中带有I/O引脚使能的外部阻容振荡
INTIO1	内部振荡器，RA6中有FOSC/4频率输出，RA7中带有I/O
INTO2	RA6及RA7中带有I/O的内部振荡器
EC	具有FOSC/4频率输出的外部时钟
ECIO	RA6中带有I/O的外部时钟

图 4-15 PIC18F4520 系列振荡器类型

晶体振荡器最为准确，它们需要在引线上连接电容来稳定振荡频率。图 4-16 给出了各种晶体振荡器的频率范围。

振荡器类型	晶体频率	典型电容值测试	
		C1	C2
LP	32kHz	30pF	30pF
XT	1MHz	15pF	15pF
	4MHz	15pF	15pF
HS	4MHz	15pF	15pF
	10MHz	15pF	15pF
	20MHz	15pF	15pF
	25MHz	0pF	5pF
	25MHz	15pF	15pF

图 4-16 PIC18F4520 晶振电容范围（经微芯科技股份有限公司许可使用）

基于电阻电容的振荡器可以代替晶体振荡器使用。该类振荡器只用两个无源元件，其成本比晶体振荡器更低。基于 RC 的时钟只适用于对精度要求不高的应用。图 4-17 给出了 PIC18F4520 的 RC 连接情况。

时钟频率是一个和电阻、电容、电压以及工作温度有关的函数，同时，也会由于制造公差的不同而出现差异。儿童电子玩具便是一个典型的应用实例，其对时钟的精度要求就不高。

### 4.3.3 复位

微控制器在上电时会将自己复位到一个固定的初始化状态。在复位过程中，会读取配置控制字，设置默认的参数值，读取第一条指令并

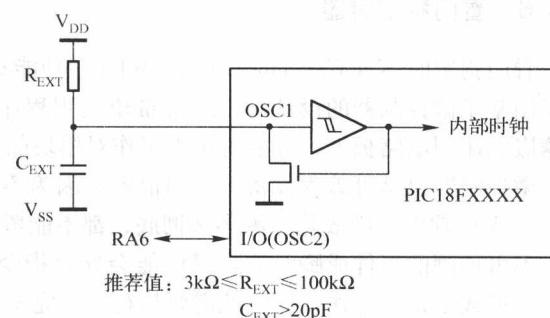


图 4-17 RC 振荡器（经微芯科技股份有限公司许可使用）

执行。根据不同的微控制器设计，可能会清除内存数据、初始化外设功能。

上电复位功能检测  $V_{dd}$  电压的升高，当达到指定的电压阈值时，微控制器逻辑将会与外部时钟源进行同步。经过一个固定的时间周期，当电压稳定在  $V_{dd}$  时，将会触发内部的复位脉冲。

大多数微控制器还支持软复位，并将其作为一条指令。执行软复位时，它会复位程序计数器，而不用将微控制器重新初始化到上电状态。这样无需修改控制寄存器文件的内容，便可直接重新启动程序。

软复位还用于多芯片应用，在上电后实现系统同步。例如，通过 I/O 端口引脚可以实现这样的功能，以保证其他系统逻辑与主微控制器进行同步。

#### 4.3.4 待机模式

对于诸如手机等低功耗应用，最大限度地延长电池寿命是至关重要的。微控制器集成了休眠模式，可以关闭所有不必要的逻辑。该模式可以通过将时钟频率降为 0 或设置成直流电压来实现。在休眠模式下，只有激活微控制器所需的最小电路处于活动状态，只消耗一点电流。

执行休眠指令将使微控制器进入掉电程序， $n+1$  条指令将被压入堆栈。休眠指令将是程序计数器停止之前所执行的最后一条指令。

如下几个事件可以唤醒微控制器。典型事件主要有芯片复位信号、指定计时器计时完成或者检测到中断。使用计时器发出强制唤醒信号可以确保该电路在没能获得中断时被激活，这样可以允许程序对中断进行检测，以判断微控制器是否处于错误状态。

例如，手机的微控制器将监控设备的活动情况，它控制着系统时钟，如果一段时间内没有操作，将会启动掉电程序，从而使手机进入待机模式；当发现有操作时，微控制器将给电路上电，并给出适当的响应。

#### 4.3.5 低功耗

在不采用睡眠模式的情况下，通过降低系统的供电电压  $V_{dd}$  可以降低微控制器的整体功耗，如图 4-18 所示。一般来说，TTL 逻辑电路的  $V_{cc}$  可以有  $\pm 5\%$  的波动，电压范围为 4.5 ~ 5.5 V。商业 CMOS 微控制器的电压变化幅度往往超过 4 V，可以从 1.5 V 变化到 5.5 V。只有更低的电压才能满足超低功耗应用要求。

微控制器类型	电压范围 (V)
Zilog Z8	2.7~3.6
Freescale 68HC11	3.0~5.5
PIC18F4520	2.0~5.5

图 4-18 微控制器操作电压范围

#### 4.3.6 看门狗定时器

看门狗定时器（Watchdog Timer, WDT）功能对于中断系统设计的鲁棒性起着关键作用。人们采用基于微控制器的设计时，通常希望应用程序可以自行工作。它们作为应用电路的一部分被镶嵌到印制电路板上，其能否正常工作对模块的整体可靠性是至关重要的。

微控制器也是可编程设备，它们很容易因为不可预知的工作条件而陷入软件缺陷（Software Bug）。无论软件全面故障检测多么彻底，都不能覆盖现实世界上所有可能的情况。

不可预测的事件或瞬时故障很可能导致指令执行顺序错误，微控制器可能会停留在一个死循环里或不正常运作，这时就需要执行一个完整的上电复位程序。

看门狗功能正是为了预防这种情况而设计的。如果 WDT 在到期之前没有重置，它将通过创建一个硬件中断来使微控制器复位，强制程序计数器跳转到中断地址，指令序列将重新正常执行。

看门狗定时器是外部中断使能，没有相应的使能位。对于芯片 PIC18F4520 系列，WDT 在芯片编程时被启用或禁止，可以防止软件由于错误执行而禁用 WDT。

#### 4.3.7 在线编程

微控制器连同其他设备模块一起焊接到 PCB 上，如果代码有错误可以改正，如果是其他的就不好修改了，将需要更换整个模块。在线编程可以在不移除系统 PCB 模块的情况下更新微控制器中的软件，该项功能在降低产品成本方面非常具有价值。

专用功能寄存器控制编程的顺序，更新后的代码转移到数据存储器，比如说，可以通过 USART 外设功能来完成控制。接下来，微控制器被置为编程模式，借助于更新的软件，一系列的指令将被执行，以实现对 FLASH 存储器进行编程。对于 PIC18F4520 的 Microchip 系列来说，可以使用与 FLASH 存储器编程有关的专用控制寄存器，有四个表控制寄存器将被用来定义在将要编程的数据存储器表中的指令地址，另外有两个控制寄存器（EECON1, EECON2）控制着编程序列的写入阶段。

### 4.4 数据输入/输出

微控制器在输入/输出上采用了灵活的方法。许多微控制器提供串行位处理，包括标准的并行 I/O 功能，并通常将 I/O 映射和内存映射功能都纳入其中。

#### 4.4.1 并行 I/O

并行端口是微控制器最基本的 I/O 功能。对于大多数微控制器来说，并行 I/O 的位宽为 1 个字节，而且每一位可以选择为输入或输出。这样既可以允许程序将其当作标准的字节接口来对待，也可以按位进行信号控制，大大提高了 I/O 接口的灵活性。图 4-19 给出了普遍使用的 NXP 公司 80C51 的 I/O 端口框图。

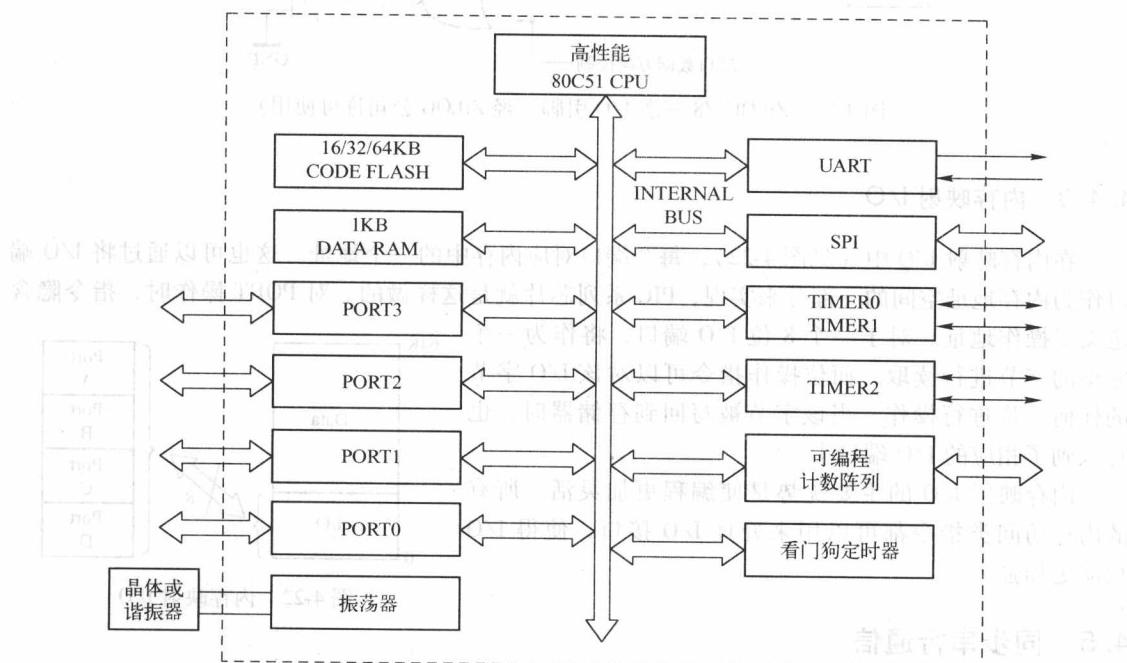


图 4-19 80C51 典型 I/O 模块框图（经 NXP 半导体公司许可使用）

作为比较, 图 4-20 列出了 ZiLOG Z8 的 I/O 引脚。该芯片采用封装类型较小, 相应端口的引脚数量也更少, 成本也更低。

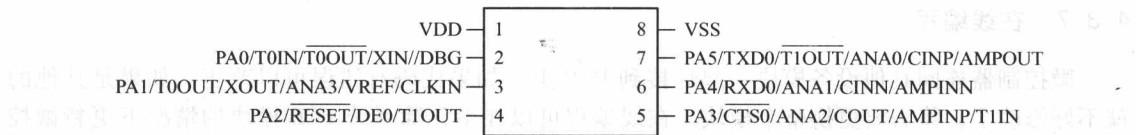


图 4-20 8 引脚封装的 I/O 引脚设计 (经 ZiLOG 公司许可使用)

#### 4.4.2 三态 I/O 引脚

I/O 引脚的电路设计技术领域引入了三态概念, 图 4-21 给出了一个使用三态 I/O 引脚的例子。输出时, 数据位被锁存到输出 D - 锁存器; 输入时, 输出驱动放大器置于高阻抗状态, 这使得输入的数据位可从输入 D 锁存器中读取。

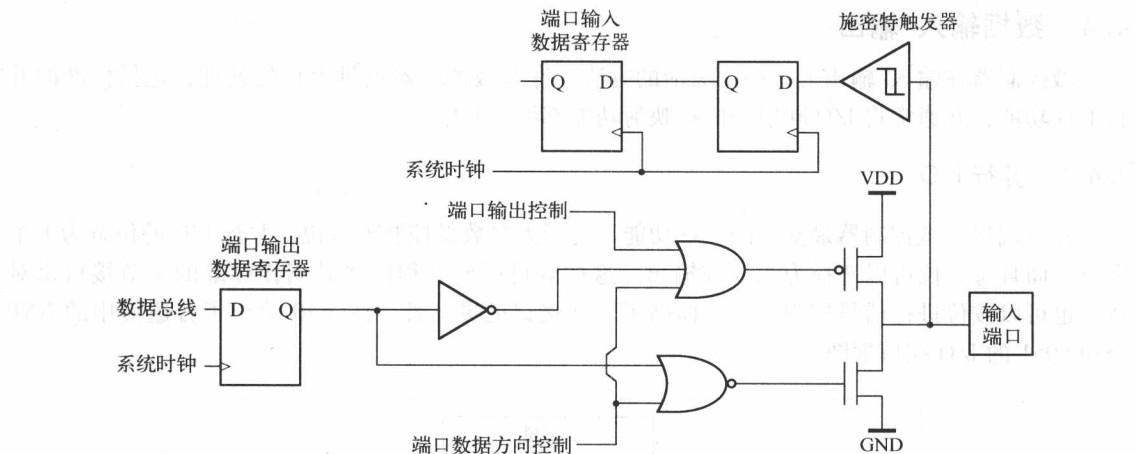


图 4-21 ZiLOG Z8 三态 I/O 引脚 (经 ZiLOG 公司许可使用)

#### 4.4.3 内存映射 I/O

在内存映射 I/O 中 (见图 4-22), 每个端口对应内存中的一个地址。这也可以通过将 I/O 端口作为内存地址空间的一部分来实现, PIC 系列芯片就是这样做的。对 PORT 操作时, 指令隐含定义了操作地址。对于一个 8 位 I/O 端口, 将作为一个完整的字节进行读取, 而位操作指令可以对该 I/O 字节的任何一位进行操作。当该字节被写回到存储器时, 也写入到了相应的 I/O 端口中。

内存映射 I/O 的主要优势是使编程更加灵活。所有的内存访问类指令都可以用来处理 I/O 接口, 使得 I/O 功能更加强大。

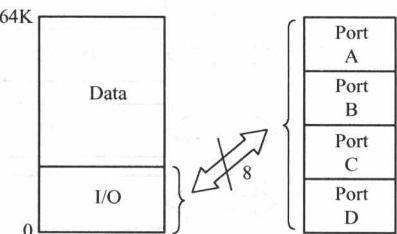


图 4-22 内存映射 I/O

#### 4.5 同步串行通信

设备之间最简单的通信形式就是串行位传输, 如图 4-23 所示。在印制电路板上, 只需要连

接两个需要连接的设备的一个引脚 ( $V_{dd}$  和  $V_{ss}$  连接在一起)。许多基于微控制器的设计都包含多个智能外设，它们之间的通信比简单的引脚连接要复杂得多。SPI 和 I<sup>2</sup>C 接口便是为了满足这一要求而专门开发的。

两个设备间实现通信，最简单的技术就是 Bit-Banging，其中一个设备，如主设备，发送位，同时从设备不断对该位进行检测；发送和接受方可以互换，在传输过程中使用很少的代码就可以处理这些位。

对于简单的设计来说，这可能是实现微控制器之间进行通信的一种较为简单的方法。然而，如果引入更多的设备，这种方法就会变得很复杂而且容易出错。有很多技术可以使得这种通信变得简单，同步串行通信就可以解决这些问题。

许多应用程序使用多个微控制器或智能外设。例如显示微控制器、A / D 转换器以及外部 EEPROM 等设备都需要快速通信的方法。从实际应用来说，并行总线并不具有成本效益，也没有必要在多核微控制器的设计中使用。高速串行接口是一个比较实用的解决办法。

有两个同步串行接口被定义为行业标准，即 SPI 接口和 I<sup>2</sup>C 接口。每一接口都为工程师提供了不同的设计方案，这些接口适用于多芯片设计，同时也可节省开销。

根据定义，使用 SPI 或 I<sup>2</sup>C 意味着已经对设计和生产费用进行了预算。这些接口的价值在于它们有能力支持复杂的微控制器设计，允许设计按照功能模块进行划分，便于管理。

## 习题

1. 为什么 CMOS 晶体管功耗比双极型晶体管低？
2. 请列举微控制设备的三种典型温度范围。
3. 请描述 DRAM 与 SRAM 的基本区别。
4. 为什么使用基于 RC 的振荡器？
5. 请问软复位与硬复位有何区别？
6. WDT 能够提供什么作用？
7. 在线编程的主要优点是什么？
8. 三态逻辑的三种状态是什么？
9. 请描述内存映射 I/O 的主要优点。
10. 同步串行通信是什么意思？

## 参考文献

- Microchip PIC18F2420/2520/4420/4520 data sheet.* 2004. 28/40/44-Pin enhanced flash microcontrollers with 10-bit A/D and nanoWatt technology. Microchip Technology.
- M68HC11E family data sheet.* 2005. Freescale Semiconductor.
- NXP (Philips), P89V51RB2/RC2/RD2 8-bit 80C51 Product Manual.* 2004. NXP Corporation.
- Z8 Encore! XP F08xA series high-performance microcontrollers with eXtended peripherals data sheet.* ZiLOG, [www.zilog.com](http://www.zilog.com).

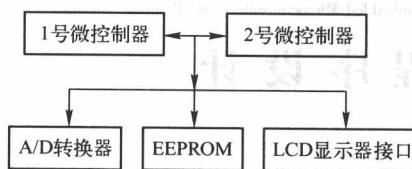


图 4-23 板卡级串行通信

## 第5章

Embedded Microcontrollers and Processor Design

# 程序设计

- 本章目标：了解基于微控制器设计的软件程序技术
- 主要内容：

  1. 轮询的概念
  2. 中断的概念
  3. 实时操作系统简介

## 5.0 程序设计

基于微控制器的设计中应用了两种不同风格的软件编程。轮询是一种基本的方法，其中程序会重复“询问”外围功能模块是否需要服务。而在中断驱动的设计中，外围功能模块会明确地“告诉”程序需要服务，如图 5-1 所示。对于应用程序来说，轮询更加简单容易一些。

有了轮询，程序总是知道何时会与外围设备交流。交流的请求会在二者之间及时同步，如图 5-2 所示，程序完全控制在外设接口交流进程之上。

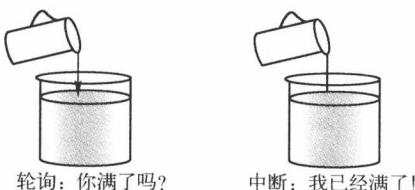


图 5-1 轮询和中断

需要运行吗？NO  
需要运行吗？NO  
需要运行吗？YES=>运行例程

图 5-2 轮询同步

## 5.1 轮询程序

简单地说，轮询只是由程序询问外围功能的状态。如果需要运行，程序会按照要求去完成，比如数据文字的传输、变量的更新以及设置一个状态位。图 5-3 展示了一个典型的轮询排列，是关于轮询的简单应用：读取键盘的输入，列出数字，然后打印出来。

从程序的时间流程上可以看出，大部分流程的时间都在等待，像键盘的按压、显示器上晶体的排列以及页面的打印。这是轮询技术的一个重要特征：微控制器把大部分时间都花在等待处理上。

### 5.1.1 程序流程

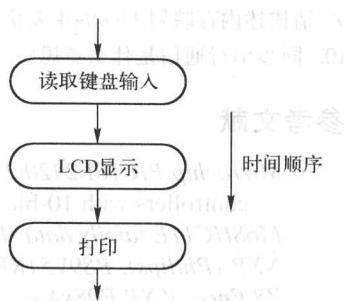


图 5-3 轮询循环

外围设备准备就绪后会设置一个标志位，程序会对这个标志位进行反复的测试，满足测试条件后，程序便开始数据传输。图 5-4 所示为对键盘、显示器、打印机的状态进行测试的流程图，这个过程非常容易实现，而且十分可靠。

图 5-5 所示为 PIC18F4520 的指令序列，每个设备的标志位都已检查完毕，并且已有一个服务开始运行。如果没有服务请求，则程序继续循环。需要注意的是程序的大部分时间都是在循环

和等待标志位的设置。

### 5.1.2 程序时序

时序是轮询技术中的一个非常关键的问题。需要花费多长时间处理服务请求决定了程序的结构(见图5-6),并且最终决定了轮询技术是否有效。处理键盘、显示器和打印机的请求需要固定的一段时间。在这段时间里,程序正在处理服务请求,因而其他的任务无法运行。

如果处理键盘请求花费的时间太长会怎么样呢?当键盘服务请求的处理时间超过了20μs,显示器的字符显示则会延迟,如图5-7所示。然而,打印机正在等着下一个需要处理的字符。那么处理键盘例程所超出的时间将会引发打印机的错误。这个字符将不会打印出来。

如果打印机这边出了问题,那么其服务时间将会延长,下一个关键的按键就有可能丢失,使用轮询时,完全掌控程序服务例程的时序显得十分必要。一个服务例程上超出的时间可能会导致其他例程的错误。

正如我们所见,在轮询程序中,微控制器多数时间都花费在等待事件发生上。随着越来越多的I/O服务例程整合到设计中,时序问题会显得更为关键,任何时序的偏差都可能引发连锁效应,从而可能诱发多种错误。



图5-6 轮询循环延时

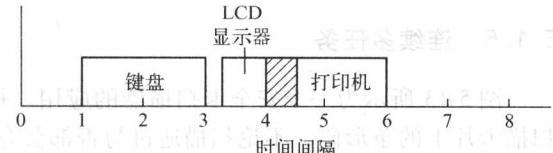


图5-7 运行时间重叠

### 5.1.3 连续任务

轮询技术可以看作是杂技中的多球戏法。杂技演员把球抛向空中的同时,手里的球也在不停转动着。如图5-8所示。

我们可以这样来描述时间图像:X轴表示时间,Y轴表示球是否在空中,如图5-9所示。正常情况下,抛起的球的间隔时间是一致的。随着球的数量的增加,时间间隔随之减小。减小到某一时间点后,一个球抛出之前就没有足够时间去接另一个球了,这个时间就是轮询循环所需的小任务时间。

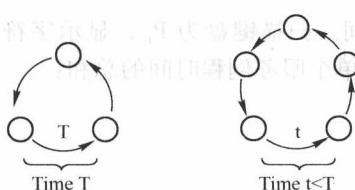


图5-8 杂技球

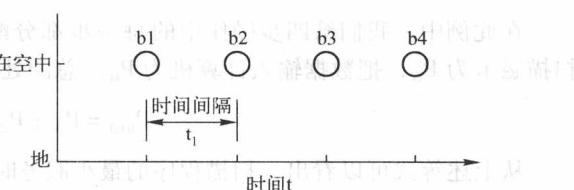


图5-9 球被抛起的间隔时间

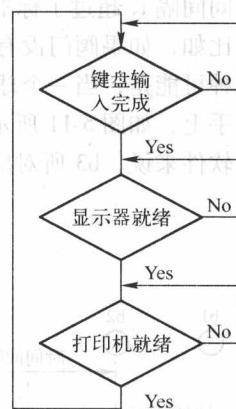


图5-4 标志位轮询

Main-Loop	btfSz	Kypd,Rdy
	goto	Kypd_service
LCD	btfSz	LCD,IRdy
	goto	LCD_service
Printer	btfSz	Printer,Rdy
	goto	Printer_service
	goto	Main-Loop

图5-5 轮询指令序列

如果杂技员掉落了一个球会怎么样呢？掉落的那个球需要花费额外的时间，在图 5-10 中可以看出时间间隔  $t_2$  超过了标准时间间隔  $t_1$ 。因此， $b_4$  号球会跑出序列外，这种情况就像软件错误一样。比如，如果阀门没有在适当的时候关闭，则会引起水灾。

另一种可能是，当一个球掉落后，在另一个球落下前，杂技员有足够的时间把球捡起来并传到另一只手上，如图 5-11 所示。如果是这种情况，那么从外部看并没有出现什么失误，一切正常。对于软件来说， $b_3$  所对应的任务将被及时处理，所有任务都会正常工作，程序也会正常运行。

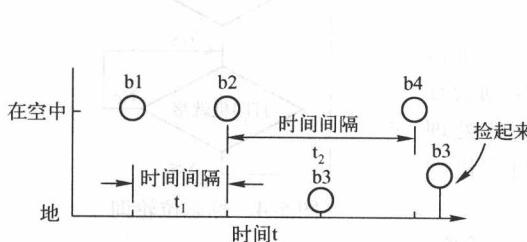


图 5-10 错过时隙

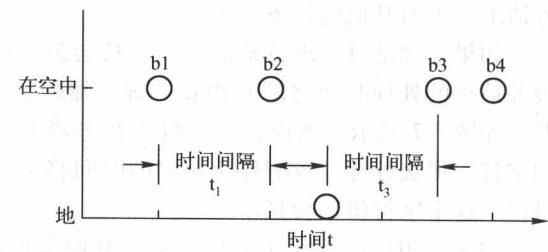


图 5-11 时隙错过补救

### 5.1.4 任务时序

在实际的设计中，任务的处理时间经常是可变的。我们生活在一个模拟的世界。就轮询而言，外围设备必须要处理前有一个最短时间（最坏情况）（见图 5-12）。换句话说，这就是微控制器能够花费在其他任务或其他更重要的任务上的最大时间（最坏的情况下）。如果超过服务间的最短时间，就会出现错误。

### 5.1.5 连续多任务

图 5-13 所示为一个安全卡扫描器的应用。扫描程序会一直循环，直到插入卡片，然后开始扫描卡片上的条形码，不论扫描通过与否都会有所显示。键盘的每一次按键同样都会进行扫描，一旦输入完成，扫描结果会发送到电脑，然后电脑会控制门闩是否打开。



图 5-12 最小任务服务时间

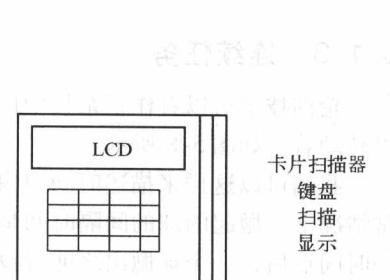


图 5-13 安全标签阅读器

在此例中，我们给四步操作中的每一步都分配处理时间。扫描键盘为  $P_{11}$ ，显示字符为  $P_{12}$ ，扫描磁卡为  $P_{13}$ ，把数据输入计算机为  $P_{14}$ 。总的处理时间为单个服务例程时间的总和：

$$P_{\text{total}} = P_{11} + P_{12} + P_{13} + P_{14}$$

从上述等式可以看出，扫描程序的最小服务时间为：

$$P_{m3} = P_{\text{total}} - P_{11} - P_{12} - P_{14}$$

如果第一张卡扫描完之后很快就扫描第二张卡将会出现什么情况呢？如果第二张卡的扫描时间低于  $P_{m3}$ ，那么它就不能被读出。这和杂技员抛丢球的情况类似。

这样的绝对时间约束是轮询技术的限制因素。外设服务之间的最短时间间隔必须小于程序所有活动的总时间。由于很难预测异步的行为，对于 I/O 外设的控制来说，轮询受到了很大的限制。

## 5.2 中断

如果杂技员在空中同时抛动 4 个球。当一个球抛起来的时候，他等着下一个球掉下来。如此重复不断，他的大部分时间都在等待球掉落。如果他的技术够好，他可以表演杂耍的同时嘴里嚼着口香糖，如图 5-14 所示。此刻，他在同时执行多个任务。

这就是中断的本质，它看起来能够同时执行多个任务。微控制器运行程序的速度相当快，但是大部分的时间仍花在等待上。如果那些时间能用于其他有效的任务上，那么不用提高电路的运行速度，系统的性能都能有很大的提高。

### 5.2.1 异步时序

中断可以引起微控制器的即时响应，转而运行中断服务程序（Interrupt Service Routine, ISR）。中断服务代码将处理这个事件，然后返回主程序。

程序执行的总时序是中断的基础。它依据的原理是某些慢的程序运行的同时，另外一些快的程序可以同时运行。就像杂技员一样，如果微处理器处理中断的时间足够快，就不会意识到有个球已经掉落了。他们会以合适的顺序继续循环下去。

有了中断，程序代码就不必约束在一个固定的线性循环流程中。程序可以更加灵活的方式来处理事件，如图 5-15 所示。程序可以通过响应事件来实现事件驱动的编程方式。基于中断的软件设计也可以称作实时程序设计。

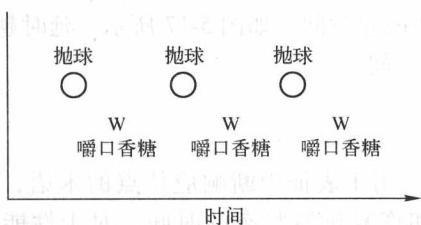


图 5-14 多任务

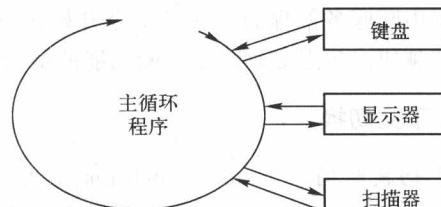


图 5-15 中断程序流程

### 5.2.2 中断允许

从定义上来看，发生中断与执行程序不是同步的。因此，程序执行期间允不允许中断发生就显得十分重要。至少，软件需要意识到中断可能会发生，万一程序正在处理重要的东西，那么中断就可能引起错误。

每个中断都有个使能位，该使能位作为逻辑与门的一个输入，位于相应的位掩码中，如图 5-16 所示。

大多数商用微控制器会有多种可能的中断源：片上的外设功能模块可能产生中断，外部也可能产生中断，其他的中断可能用于支持软件代码调试。

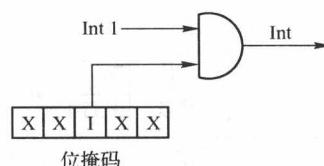


图 5-16 中断允许

每个中断都有相应的使能位。此外，通常还包含一个全局的中断使能位，如果将该使能位置零，则屏蔽所有中断。对于有总中断源的微控制器来说，可能会出现附加的外设使能位，就像 PIC18F4520 微芯片系列中的一样。

没有相应使能位的中断称为不可屏蔽中断，意味着它们不能通过软件来关闭。没有指令可以屏蔽此类中断。如果执行出现错误将所有中断屏蔽了，那么微控制器可能陷入死循环。此时，非屏蔽中断将使程序强制退出死循环，返回运行模式。

### 5.2.3 机器状态

当中断发生时，微控制器将会处于一种特定的逻辑状态。这种状态会随着微控制器类型的不同而不同。寄存器和状态字将包含当前进程的特定数据，执行中断代码前，中断服务的一部分会保存处理器的关键数据。服务例程的最后，机器状态会被复原。

当中断发生时，最重要的任务就是保存机器状态。当机器状态恢复时，指令序列可以重新执行，这样看起来就像没有发生中断一样。这就要求对机器状态进行定义。

最简单的做法是，与当前指令执行相关的所有值都必须保存下来，因为中断可能影响专用寄存器中的内容和标志位。指令执行完成前，这些值时时刻刻都有可能改变。

那些会被任何后续指令直接修改的数值必须在中断前保存下来。一旦中断发生，处理器硬件会将程序计数器修改到新指令地址。第一批顺序指令序列必须在被修改前保存下来，这是重要的数据。

并不是所有情况下都需要保存机器状态。但是，这么做是一个非常好的编程习惯。除了对性能要求极高的应用，处理少量保存类型的指令并不会对性能造成多大影响。比如，在 Microchip PIC18F4520 中，只有 STATUS、W 和 BSR 寄存器需要保存。

### 5.2.4 延时

从检测中断直到合适的软件例程开始执行所花的时间称为延时。延时一般理解为从接收到中断开始到中断服务例程的第一条指令开始运行之间的总时间，如图 5-17 所示。延时越短，中断的响应就越快。延迟必须小于中断功能的最大服务时间。

### 5.2.5 上下文切换

上下文切换时间（context switch time）是另一个用于表征中断响应特点的术语，它是指软件例程之间转换控制所需要的时间，包括保存和恢复机器状态的时间。对于性能要求非常高的微控制器应用，比如千兆交换机，这是一个系统性能的限制参数。除了状态位之外，相当数量的寄存器内容也必须保存下来。一种解决方法是配备更多的寄存器，那样处理器只需要简单地重新映射寄存器寻址，从而省去了保存和恢复的操作。如图 5-18 所示为一个寄存器组的地址指针。

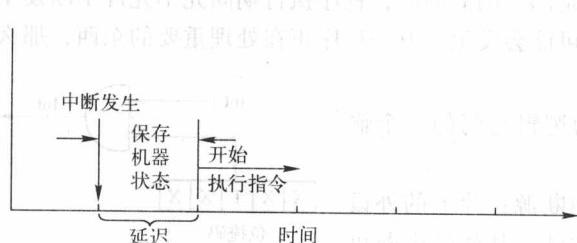


图 5-17 中断延迟时间

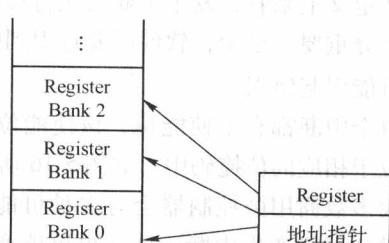


图 5-18 寄存器组

### 5.2.6 中断向量

程序控制通过如下三种方式转换到中断服务程序上，如图 5-19 所示。

- 单个中断向量。
- 中断向量表。
- 可编程的中断向量。

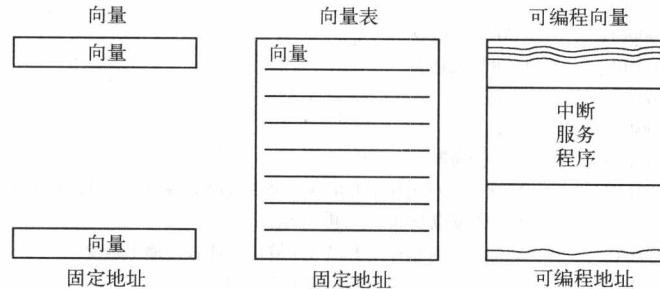


图 5-19 中断向量

Microchip PIC18F4520 只有两个中断向量，高优先级的中断位于 0x0008，低优先级的中断位于 0x0018。当中断发生时，下一个顺序地址  $n + 1$  被压入程序地址栈，中断向量中的 GOTO 指令将程序跳转到中断服务程序中，如图 5-20 所示；在中断服务程序的结尾，RETFIE（返回）指令弹出堆栈，并返回  $n + 1$  的位置继续执行。

ORG	Ox0008	; 高优先级中断向量
GOTO	Int-Service-High	; 转到中断服务程序
:		

图 5-20 基于向量的中断

Microchip PIC18F4520 有多种可能的中断源。中断服务程序的第一个任务就是判断中断源，然后跳转到相应的中断服务程序。这个过程需要花费一段时间，如果大量的中断正在运行，那么指定中断所需的响应时间就成了问题。这是限制系统性能的一个方面，同时也可能成为选择微控制器的决定性因素。

当中断发生时，微控制器硬件会将中断服务程序地址载入程序计数器。此地址可以是对应所有中断的一个固定内存地址，比如中断向量表或者 Microchip PIC18F4520 中所采用的。与大多数的微控制器一样，ZiLOG eZ8 也采用了中断向量表，如图 5-21 所示，表内包含了 20 个中断服务程序的地址。例如，TIMERO 中断将指向地址 0x000C。

不同于单个中断向量，该技术提供了更为快速的响应。任何中断都可以编入到三层优先级中的一层。这无疑增强了针对特定应用的能力，而缩短了响应时间（延迟）。

### 5.2.7 中断嵌套

在含有多个中断的系统中，很有可能当程序在执行 ISR 时出现一个新的中断。在这种情况下，这些中断会发生嵌套，如图 5-22 所示。这是由中断出现数量和级数决定的。

当出现多个中断时，地址栈就用来存储返回地址。在每个中断发生时，都会把返回地址压入栈中，中断嵌套的数量受到地址栈深度的限制。在基于栈的结构中，子例程也可以调用指令。指令将返回地址压入栈中或者弹出栈。

中断优先级	程序存储器 向量地址	中断源
最高	0002H	复位（非中断）
	0004H	看门狗定时器（见看门狗定时器章节）
	003AH	主振荡器错误陷阱（非中断）
	003CH	看门狗定时器振荡器错误陷阱（非中断）
	0006H	非法中断陷阱（非中断）
	0008H	保留
	000AH	定时器1
	000CH	定时器2
	000EH	UART0接收器
	0010H	UART0发送器
	0012H	保留
	0014H	保留
	0016H	模数转换器
	0018H	A7端口，可选择上升沿或下降沿或LVD触发（参见21页关于复位、停止模式恢复以及低压检测的介绍）
	001AH	A6端口，可选择上升沿或下降沿或比较器输出触发
	001CH	A5端口，可选择上升沿或下降沿触发
	001EH	A4端口，可选择上升沿或下降沿触发
	0020H	A3或D3端口，可选择上升沿或下降沿触发
	0022H	A2或D2端口，可选择上升沿或下降沿触发
	0024H	A1端口，可选择上升沿或下降沿触发
	0026H	A0端口，可选择上升沿或下降沿触发
	0028H	保留
	002AH	保留
	002CH	保留
	002EH	保留
	0030H	C3端口，上升与下降边沿都触发

图 5-21 ZiLOG eZ8 中断向量表（经 ZiLOG 公司许可使用）

堆栈与缓冲区不同，缓冲区是一个先入先出的随机存储器（Random-Access Memory，RAM），而堆栈是一个后入先出的 RAM，如图 5-23 所示。这些指令将地址压入堆栈的顶部。下一个返回操作会弹出堆栈内容，转移程序执行。程序员必须确保压栈和出栈的协调，以避免堆栈溢出或下溢。

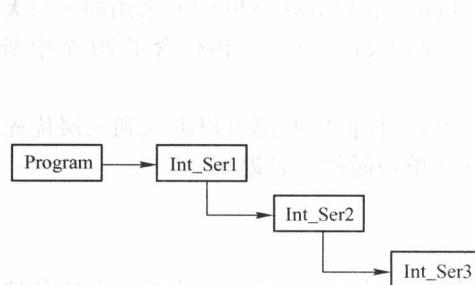


图 5-22 中断嵌套

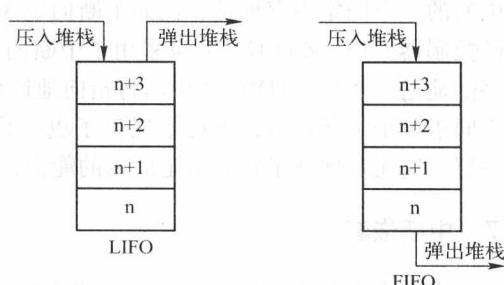


图 5-23 LIFO 和 FIFO 堆栈

## 5.2.8 关键代码

有时允许中断并不可取。当进程正执行关键代码时，中断可能会带来不利的结果。比如医疗

器械在控制心跳的时候是不希望产生不可预测的中断的。

用户可以对中断标志进行轮询，可通过一个工作于延迟模式的定时器来实现。因为程序要等待定时器计时完成，而定时器计时结束后会产生一个中断，因此，程序只需简单的监视定时器中断标志即可。这样，没有实行中断服务例程的程序会节省程序代码。

### 5.2.9 中断服务程序

图 5-24 中列举的程序代码显示了 Microchip PIC18F4520 中的 W、STATUS 和 BSR 寄存器是如何保存的。首先，它们被放入临时的内存地址中，当它们被保存后，中断服务程序就可以继续执行了。在程序结尾，W\_TEMP、STATUS\_TEMP 和 BSR\_TEMP 会被恢复，这样就可以在中断的同时保存机器状态。

```

Int_service
    MOVFF W_TEMP           ; copy w to temp register
    MOVFF STATUS, STATUS_TEMP ; save status register
    MOVFF BSR, BSR_TEMP      ; BSR located anywhere
    clrf PORTB             ; clear PORTB to show interrupt
    movlw h'20'              ; Green, RD<5>
    movwf PORTD             ;
;

    MOVFF BSR_TEMP, BSR      ; Restore BSR
    MOVFF W_TEMP, W          ; Restore WREG
    MOVFF STATUS_TEMP, STATUS ; Restore STATUS
;
;
; don't forget to reset the interrupt flag for RB0/INT
;
    bcf INTCON, INTF        ; set INTF to zero
    retfie                  ; return with interrupts enabled

```

图 5-24 中断服务程序实例

## 5.3 实时操作系统

实时操作系统（Real-time Operating System, RTOS）是一种服务于实时应用程序的操作系统。像 Microsoft Windows 操作系统一样，实时系统为应用软件程序和标准接口（比如与终端和打印机的连接）提供了基本架构。像 Windows 一样，实时系统通过让程序员关注应用程序代码而大大降低了编程所耗费的精力；不同于 Windows 的是，它支持程序的实时执行。

在一个有少量中断源的简单应用程序中，主程序可能使用单一指令进行简单循环（见图 5-25）。当中断发生时，程序将转移到中断向量所在位置，并从那里开始执行指令。在中断服务程序结束时，程序返回调用例程。



图 5-25 单指令的中断循环

## 5.4 事件驱动系统

实时操作系统不停地执行代码以等待中断的出现。从定义上来看，应用程序的复杂性必须满足 RTOS 的需求。卫星控制系统就是个很好的例子。RTOS 时刻在等待中断，并且准备执行中断（实时），如图 5-26 所示。

拥有鲁棒中断结构的复杂应用需要更为复杂的循环。RTOS 提供了全系统的服务，像打印机输出或终端输入这样

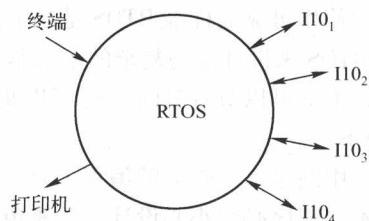


图 5-26 RTOS 系统中断

的常用任务都可以通过 RTOS 实现。

## 5.5 内核

RTOS 的关键是内核或者核心，是系统工作所需的最小限度的程序代码。应用程序和系统代码在内核的外层，如图 5-27 所示。

内核必须常驻在存储器最底层，它需要直接和处理器逻辑进行交流，以即时处理中断请求。在以最小化内存为主要目标的应用程序中，使用规模最小的内核要好一些。

RTOS 简化了软件程序任务。在诸如控制卫星之类的复杂应用程序中，商用 RTOS（如 WindRiver 公司提供的 VxWorks）提供了可靠的操作系统平台。基本 I/O 例程已经规范化，比如，通过 COM1 串行链路与 PC 进行的交流是通过 USART 来完成的。

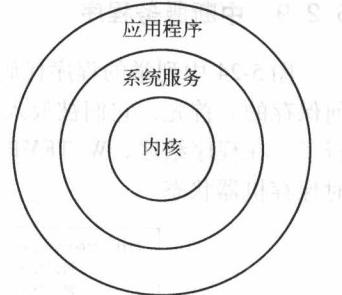


图 5-27 RTOS 分层

## 5.6 系统分层

带有关键时序约束的专用 I/O 功能模块通过专用接口直接与内核交流，如图 5-28 所示，如果将内核直接暴露给程序代码，会增加风险，所以必须要以一种绝对可控的方式来操作。

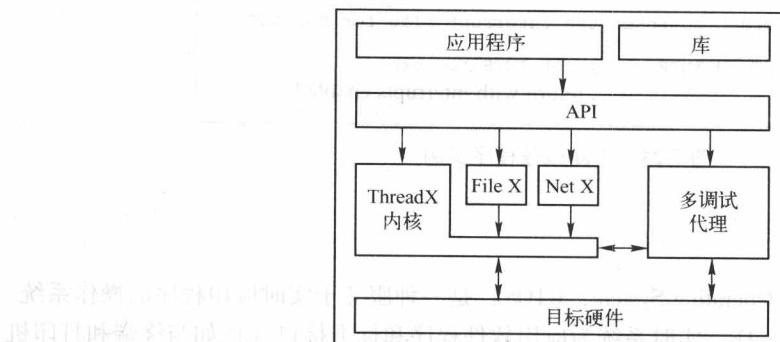


图 5-28 ThreadX 内核（经绿色希尔斯软件股份有限公司许可使用）

## 5.7 风险

基于微控制器的实时应用程序操作通常面临着失败的风险。一些不可预计的异步事件的组合可能会导致系统关闭。故障恢复是系统设计中一个重要的方面，健全的微控制器设计是具备容错能力的，它能承受不可预测的错误，可以修复并继续运行。对于卫星这样的远程系统来说，这是对设计师的一个巨大挑战。

使用可靠的标准 RTOS 是一个非常棒的设计决定，比如 WindRiver 公司的 VxWorks 系统。使用 RTOS 来设计更为复杂的系统是个非常不错的选择。对小型的应用程序来说，使用商用实时内核，不但可以节省时间，还可以提高代码的可靠性。然而，也并不是所有的应用都需要商用 RTOS。

中断是一个强大的编程工具，可以显著地提高微控制器设计的性能，但也确实会增加一些成本。程序必须小心设计，以避免不可预测的错误发生。稳健的设计技术和成熟的设计规则使设计工程师能够在合理的预算内制造出复杂的可定制实时应用程序。

## 习题

1. 描述轮询和基于中断的设计之间的主要区别。
2. 轮询程序的主要功能是什么？
3. 在一个中断驱动的设计中，标志位由什么设定？由什么解除？
4. 为什么说时序是轮询的关键？
5. 如果错过一个时钟隙会怎么样？
6. 描述最坏情况下的时序设计。
7. 在一个设计中，如果  $P_{\text{total}} = 15 \text{ ms}$ ,  $P_1 = 4 \text{ ms}$ ,  $P_3 = 3 \text{ ms}$ ,  $P_4 = 5 \text{ ms}$ ,  $P_2$  所允许的最大时间是多少？
8. 中断最基本的优点是什么？
9. 描述什么是事件？
10. 为什么需要一个中断使能位？
11. 给出一个中断屏蔽的例子并进行描述？
12. 什么是机器状态？
13. 在中断驱动的设计中，延迟指的是什么？
14. 举例说明什么是中断向量。
15. 描述中断是怎么影响嵌套的。
16. 举出一个 ISR 的例子。
17. 在 PIC18F4520 中，哪三个寄存器定义了机器状态？
18. 在基于 RTOS 的设计中，分层代码是什么意思？

## 参考文献

Joseph, M. 2001. *Real-time systems specification, verification and analysis*. Pune, India: Tata Research Development & Design Centre.

*MicroChip PIC 18F4520 data sheet*. July 2007. Microchip Technology.

Moore, R. 2001. *How to use real-time multitasking kernels in embedded systems*. Costa Mesa, CA: Micro Digital Associates, Inc.

*Thread X data sheet*. 2008. Green Hills Software, Inc.

*Why is a different operating system needed?* Symbian White Paper, Symbian. October 2003.

*ZiLOG Z8 Encore! XP F08xA Series with Extended Peripherals*. Product specification, ZiLOG. June 2006.

## 第6章 |

Embedded Microcontrollers and Processor Design

# 软/硬件调试

- 本章目标：通过 COTS 和 SoC 微控制器设计，了解软/硬件调试的过程
- 主要内容：

- IDE 与 ICE 的概念
- 实时调试
- 调试步骤
- COTS 和 SoC 调试工具

## 6.0 软/硬件调试

完成嵌入式微控制器产品设计一般需要通过三个主要的步骤。软件和硬件都需要调试，尤其要对集成于硬件中的软件进行调试，如图 6-1 所示。针对调试周期中的每一阶段都有相应的调试工具套件。

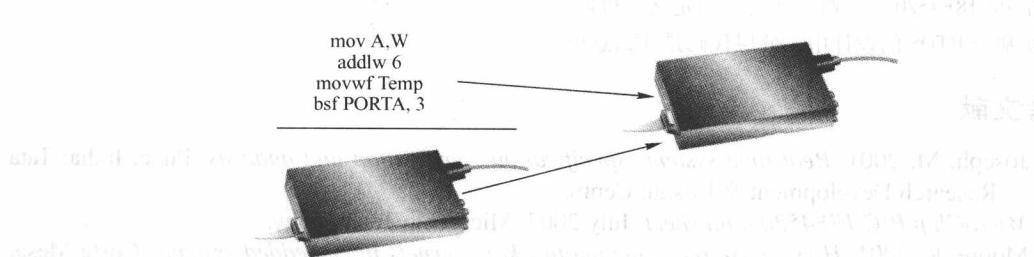


图 6-1 软/硬件调试

花费在调试过程中的时间可能会超过软件开发或硬件设计所用的时间。对设计进行调试是设计过程中固有的一个环节。对基于商用成品（commercial off-the-shelf, COTS）的应用来说，对设计的调试是经过精心定义的反复验证过程。对嵌入式微控制器来说，软/硬件调试更为重要，一般来说，花费在第一款能够正常工作的芯片上的成本都在上百万美元或更多。

## 6.1 COTS 控制器工具

新的工具软件一直在追求让调试过程变得更加高效。对于商用 COTS 微控制器来说，其结构具有明确的定义（如 i8051, HCS12, PIC18F4520），已经有许多工具软件被广泛应用于该领域。在集成开发环境（IDE）方面已经有了很大的创新。但是，如何确定软件的稳定性仍然是一个十分重要的问题。

IDE（见图 6-2）支持不同厂家的芯片系列越灵活，在市场上就越受欢迎。对基于商用 COTS 微控制器设计来说，芯片的成本很明确，并且前期的设计成本较低，IDE 的性能便显得尤为重要。在线调试器是 IDE 的扩展，它将软/硬件集成开发环境扩展到了目标板的实时调试，对于瞬时错误的定位很有用处。

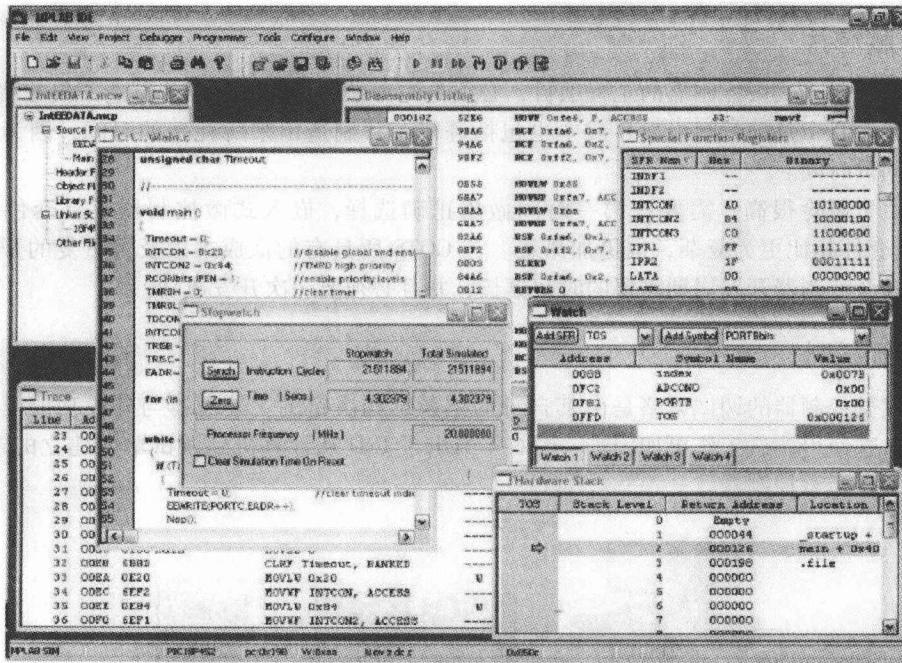


图 6-2 Microchip MPLAB IDE（经微芯科技股份有限公司许可使用）

## 6.2 嵌入式控制器工具

当前，嵌入式微控制器技术对于软/硬件的调试较为困难，因为微控制器内核位于集成电路芯片内部，不能直接接触和访问。现在已经建立了专用的内核访问机制，ARM 和 MIPS 都定义了嵌入芯片内核上的调试模块，可以访问内核状态，如图 6-3 所示。

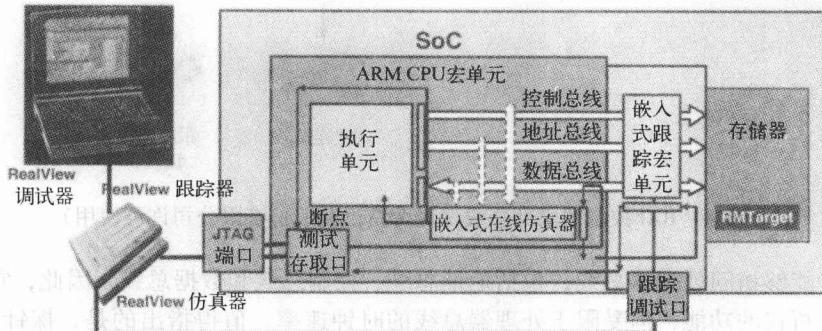


图 6-3 针对 ARM 内核的 Multi-ICE（经 ARM 有限公司许可使用）

嵌入式内核调试组件属于软件调试的较高层次，与用于 COTS 的微控制器设计类似，但更为复杂。从定义上来说，嵌入式微控制器设计比基于 COTS 的设计在成本上要高几个数量级。由于会面临如此高的经济风险，相应的工具套件也就显得更为重要，花费在这些工具上的成本也会更高。

对于简单的嵌入式内核应用，如安全证书阅读器，还需要计算花费在首款芯片上的成本。COTS 类型的软件开发工具套件可能已足以满足程序开发要求，针对过程的调试才是最为迫切的。

### 6.3 首款芯片

针对基于嵌入式微控制器的工程项目，必须首先考虑芯片问题。芯片制造不仅花费很高，而且可能会花费几个月的时间。项目成本也包括等待芯片制造出来所耗费的时间，因为这样会推迟产品的问世时间。

正是由于风险很高，需要在第一时间做出正确选择，嵌入式微控制器工具套件与相应的COTS工具套件相比更为复杂，也更加昂贵。与COTS所具有的低成本以及可重复的开发周期不同，嵌入式微控制器开发周期需要的时间更长，并且必须要一次开发成功。

### 6.4 板级探针

嵌入式微控制器的调试策略是由其自身的体系结构决定的。处理器与外设功能模块都嵌入芯片内部，这使得调试工作更加困难。图6-4给出了ICD从计算机到调试单元建立的连接关系，其中包含了目标微控制器。

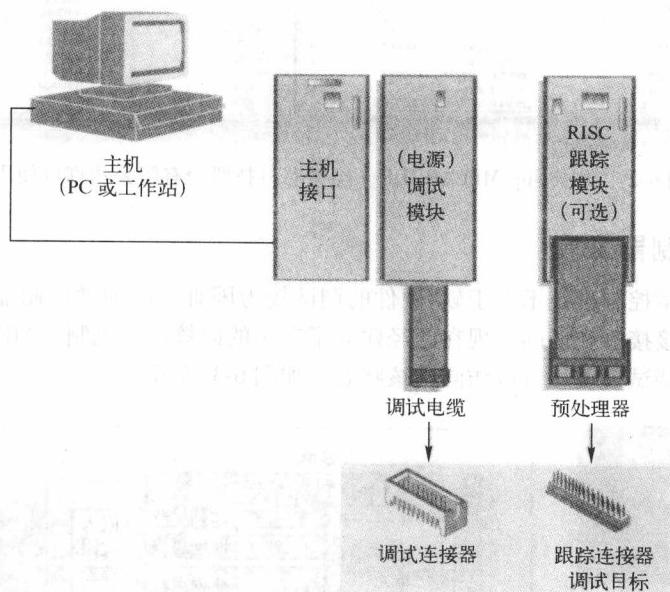


图6-4 ICD控制器内部总线（经劳特巴赫股份有限公司许可使用）

在板探针能够访问处理器总线，包括地址总线、控制总线和数据总线，因此，它们可以执行类似于逻辑分析仪的功能，但受限于处理器总线的时钟速率。值得指出的是，探针是非介入的，也就是说，探针不会对软件程序的执行产生影响。它监视总线的状态，并且可以根据调试软件定义的事件进行触发，触发条件可以是指令、存储器访问或者数据值。Microchip MPLAB ICD 2 在线调试器的主要特性如下：

- 连接主PC的RS-232串口或USB接口(2Mbit/s全速)。
- 实时后台调试。
- MPLAB IDE的图形用户界面。
- 内置过压/短路监测。
- 从PC升级固件。
- 全封闭式。

- 支持 2.0V 低电压（电压范围为 2.0~6.0V）。
- 诊断 LED（Power、Busy 和 Error）。
- 读/写内存空间和目标微控制器中的 EEDATA 数据。
- 程序配置位。
- 程序内存空间擦除及验证。
- “Freeze-on-halt” 功能允许外设在断点处停止计时器。

## 6.5 调试步骤

进行设计与调试有如下 5 个基本步骤：

- 程序输入。
- 汇编/编译。
- 程序生成。
- 软件仿真。
- 软/硬件验证。

图 6-5 给出了带有 ICD 选项的 Lauterbach TRACE32 的工作界面。TRACE32 使用一个包整合了设计调试的所有五种工具。由于这些工具都是用统一的用户界面，因而提高了工程师使用软件的效率。图 6-5 显示了高级语言调试器，其调试窗口已经打开。

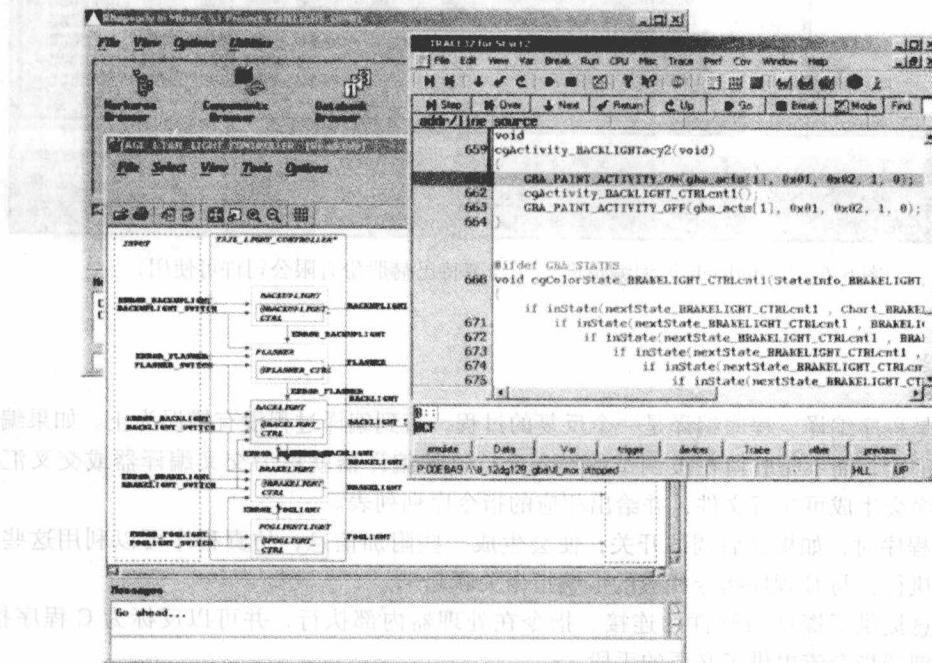


图 6-5 Lauterbach TRACE32 调试组件（经劳特巴赫股份有限公司许可使用）

对基于 COTS 的应用，IDE 的可用性成为选择微控制器的首要因素。当芯片批量很高时，芯片成本基本维持不变，产品的开发成本便成为十分重要的问题。好的 IDE 可以节省编程时间，提高调试效率，并为下游产品提供支持。

### 6.5.1 软件编辑

程序输入可使用编辑工具来实现。编辑工具可以简单如笔记本，也可以更复杂，为代码书写

提供专门功能。

输入每一条语句时，软件编辑器可以检查语句的语法。软件编辑器的输出可以是 C 代码的源文件格式（.src）或者汇编语言文件格式（.asm），如图 6-6 所示。

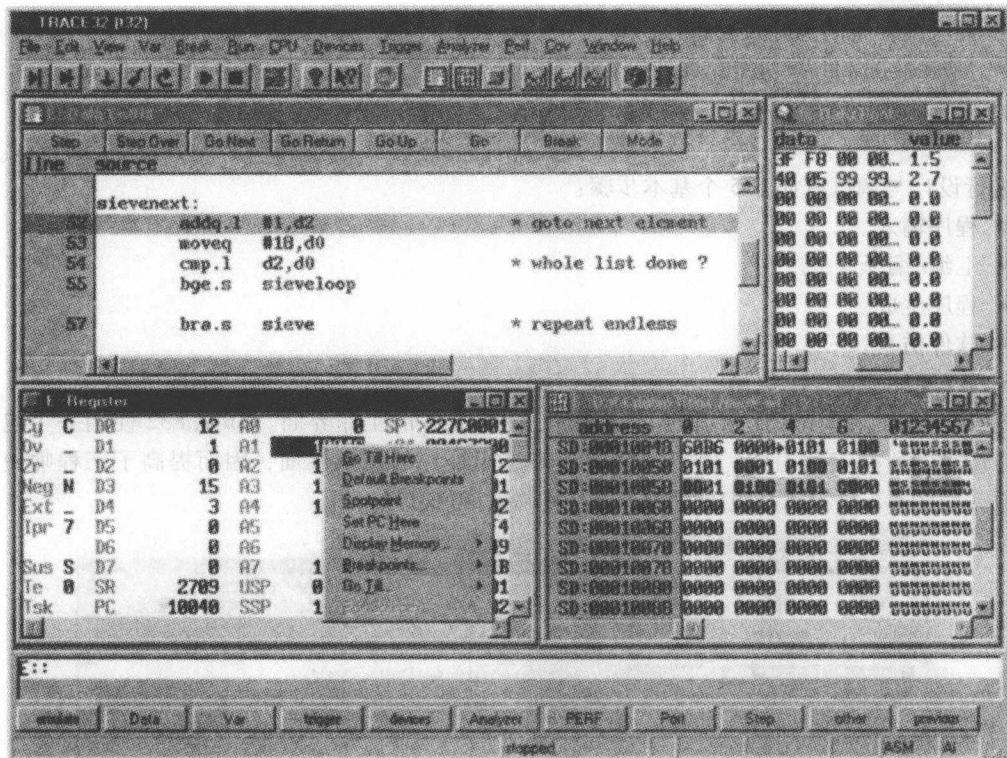


图 6-6 Lauterbach 汇编编辑模式（经劳特巴赫股份有限公司许可使用）

### 6.5.2 编译

下一步是程序编译。程序编译是一个反复的过程，直到编译过程没有错误为止。如果编译程序驻留在电脑上，而不是驻留在被调试的设备上，这种编译器就称为交叉编译器或交叉汇编程序。编译程序会生成可执行文件，并给出相应的指令序列列表。

编译 C 程序时，如果开启调试开关，便会生成一些附加信息，仿真程序可以利用这些信息跟踪指令的执行，与 C 程序指令相应的汇编指令关联起来。

调试信息提供了探针与软件的连接。指令在处理器内部执行，并可以反标为 C 程序指令。这为跟踪处理器指令流提供了必要的手段。

### 6.5.3 程序生成

程序生成 (Build) 是将软件模块合并成一个单一的程序映像。除了汇编/编译外，大多数 IDE 都包含程序模块，以简化程序生成过程。MPLAB 包含了一些文件，这些文件列出了专用文件的清单以及与每一个处理器相对应的位，这些都简化了编程任务。

对于较大的程序，建议将程序划分为不同的功能模块，这样能简化调试的过程，也能够建立一个  
多程序并行工作的机制，节省软件开发的时间，如图 6-7 所示。

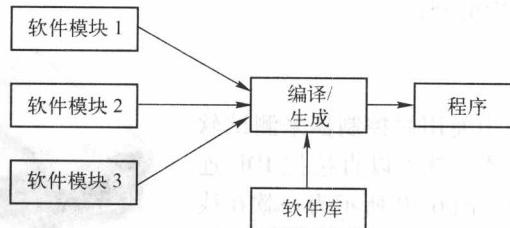


图 6-7 程序生成过程

在编译过程之后需要添加 Build 步骤。链接器的输入是程序模块和所包含的文件，更加可靠的编程也可将系统库纳入其中。Build 步骤的输出是可执行文件，也是将要下载到微控制器内存中的文件，如图 6-8 所示。

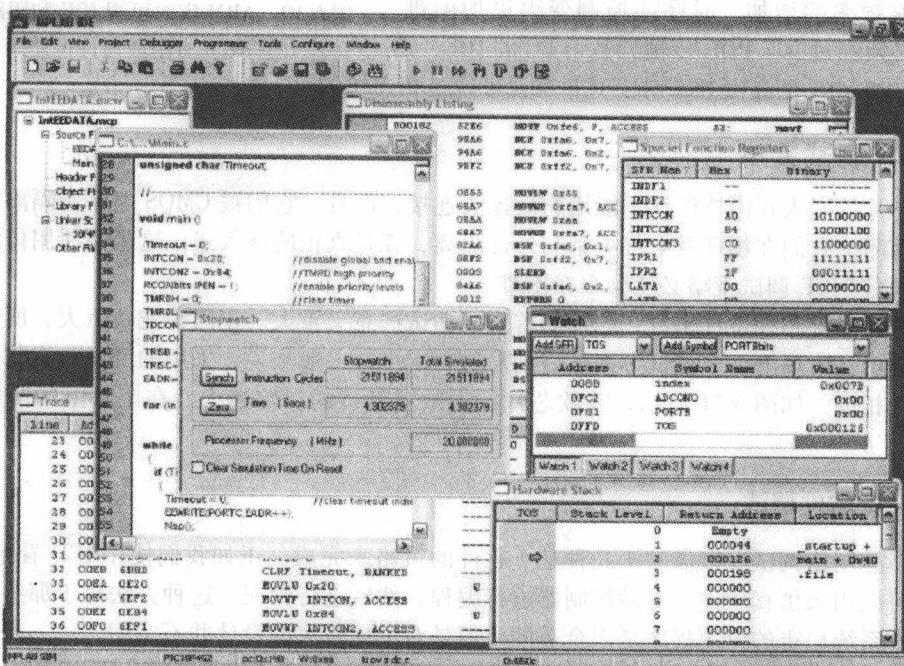


图 6-8 Microchip MPLAB 十六进制列表（经微芯科技股份有限公司许可使用）

#### 6.5.4 仿真器

程序通过编译后，下一步就是调试逻辑错误，这个功能可以通过软件仿真器来完成。仿真器是一种程序，支持目标处理器指令的执行。

仿真器调试的逻辑代码可以不需要驻留在微控制内存中便可以完成仿真，它的一个主要特点是能够支持输入激励。如果一个输入端口扫描到有键被按下，仿真器很有可能创建一个由 1 和 0 组成的数据字，来模拟如果有键按下时的输入。

用外部测试向量对软件进行测试是仿真的主要特征。测试向量在形式上是一个十六进制的表格，这个表格可以用 Excel 来实现并作为端口的输入，如图 6-9 所示。更为强大的是，它们可以是测试所有输入源的文件。这样便可以通过程序代码在

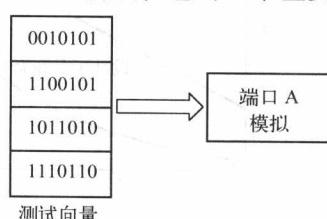


图 6-9 测试向量

仿真器中的执行对其进行全面的测试。

### 6.5.5 在线仿真

最后一步是在目标应用中使用微控制器来测试软件。采用商用 COTS 微控制器，就可以直接把 IDE 连接到印制电路板上的插座中。图 6-10 所示为一款在线仿真器（in-circuit emulation, ICE），微控制器芯片插入有探头的插座上，此插座再插入 PCB 上的微控制器插座中。

ICE 有电气和物理上的限制，它不能嵌入系统芯片的内核中。COTS 微控制器也同样受到限制。随着微控制器晶振频率的增加，时序的控制变得更加困难，总线信号必须穿过 ICE POD 传输到基于 PC 的 IDE，然后再返回。20 MHz 的晶振频率对于 ICE 单元来说是相当快的。

## 6.6 SoC 调试策略

系统芯片的嵌入式内核设计不能用物理探头连接，而且，它们被 CMOS 电路所隔离，无法从外部进行探测。它们在物理及电气上都与外部隔离，算是真正的嵌入式。这表明使用传统的商用 COTS 微控制器 ICE 调试策略必然会产生困难。

对于 SoC 系统芯片设计来说，芯片的首次成功试产意义重大。正因为花费巨大，所以首款芯片必须要成功，市场上许多调试方案正好满足这一需求。一般来说，采用的调试方案与项目的设计成本直接相关，如图 6-11 所示，首款芯片的失败对经济的影响越大，在工具上的前期投资就应该越多。

### 6.6.1 SoC 软件调试

实现微控制器调试可以借助于几种基于软件的策略来完成。正如我们使用 ICD 看到的一样，该方法也可以用来仿真软件，对微控制器进行编程，然后进行验证。这种方法对于那些只有少量代码和硬件系统稳定的应用程序是很合适的，但是对于比较大的设计并不实用。

程序仿真的另一种可供选择的方法是在代码中使用大量 C 语言 PRINTF 语句，如图 6-12 所示。PRINTF 语句的插入就如同轨道上的标记一样，如果软件的执行顺序是按照先前标记好的路径走，那么 PRINTF 将会在终端显示消息“*I am here*”。

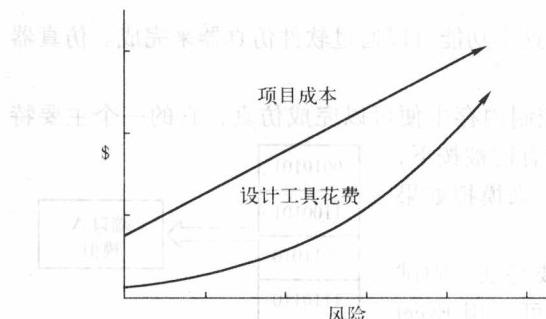


图 6-11 调试工具成本

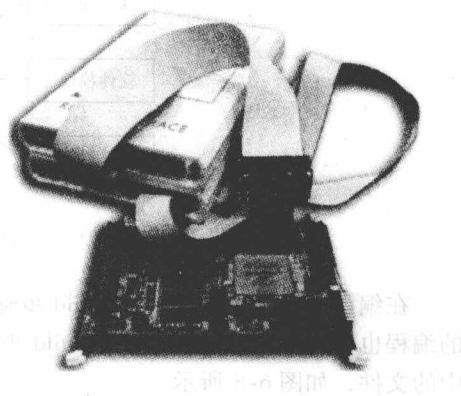


图 6-10 ARM 实时可视 ICE 和跟踪模块  
(经 ARM 有限公司许可使用)

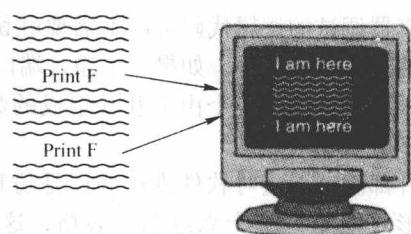


图 6-12 PRINTF 的使用

使用 PRINTF 语句的问题在于如果没有在标记 1 和标记 2 之间插入该语句将不知会发生什么。在这种情况下，应该在靠近认为是错误指令执行的地方加上 PRINTF 语句。通过不断尝试和更正错误，将会不断减少出现问题的代码段。对于规则程序该方法十分有用，但对于实时应用的复杂代码，系统将会被打断。

添加 PRINTF 代码并执行相关指令也可以改变指令流程和时序，单是这一点就能对复杂的算法时序产生影响，导致错误的产生。往 PRINTF 语句中加指令和执行它相关的指令都能改变指令流程及其时序。仅此就能够交互影响复杂时序的算法，然后产生错误。使用 PRINTF 插入到软件和硬件的实时交互中，这在实时应用中是一种插入式调试策略。

### 6.6.2 内核调试

对于 SoC 设计，非介入监视功能是必须的。瞬间错误经常发生在代码运行后的几个小时或几天后。数字有线机顶盒就可以作为一个实例。潜在的错误状态可能涉及一组很少发生的条件组合，非介入调试能够监测处理器状态，并能在这些少见的条件下产生中断。

仿真器应当具有覆盖广泛的测试向量文件。在仿真期间，仿真器不可能模拟所有可能的硬件和软件事件的组合。仿真只是一个工具，不是全能的解决方案，它缺少察觉实时与瞬间错误的能力。

在实时系统设计中，要注意的另外一点是事件的实时性，需要采取一些方法来捕捉并记录这些实时事件。从某种方式上来说，它与量子系统很相似，任何一种在测量上的尝试都能改变它的状态。对于处理器来说，需要找到一种方法在不改变其实时性的基础上能够跟踪并记录将要发生的事件。

解决方案之一是将内核调试作为 SoC 设计的一个功能模块。这样，该模块就是处理器核和 SoC 的一个集成功能模块，与其他逻辑一起运行，同时能在不打断实时流程的情况下监测处理器的活动。为此，ARM 定义了一个 EmbeddedICE 宏单元，如图 6-13 所示。

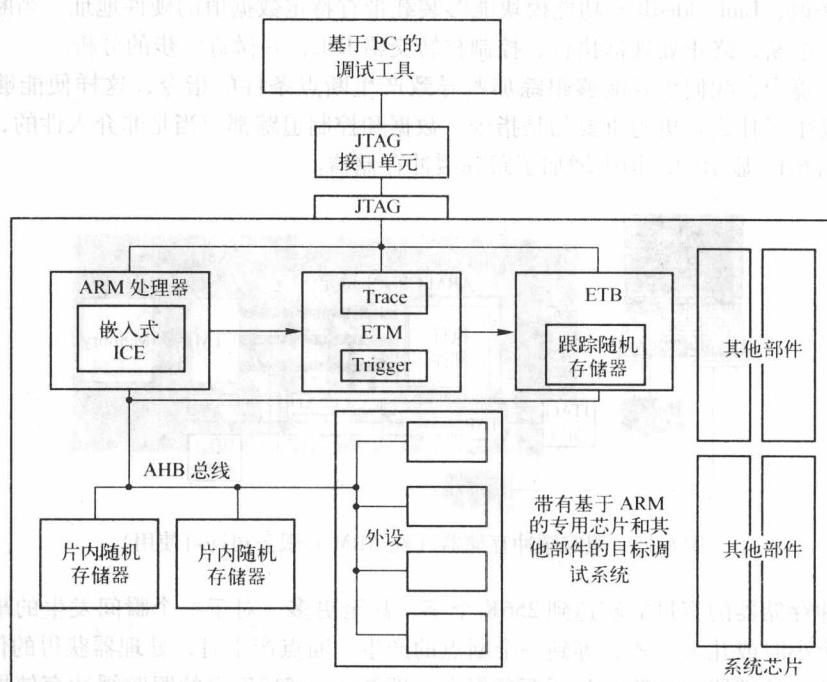


图 6-13 ARM EmbeddedICE 宏单元（经 ARM 有限公司许可使用）

### 6.6.3 JTAG/EJTAG 规范

ARM 和 MIPS 是两家主要的嵌入式内核的设计厂家，两家都开发出了各自的功能调试模块，都依赖于 JTAG/EJTAG 工业标准和测试存取口（Test Access Port, TAP）标准，如图 6-14 所示。这是一些测试标准，用于提供测试安装于 PCB 板上的集成电路的方法，实现测试的目标，是事实上的工业标准。ARM 采用 JTAG，MIPS 采用 EJTAG，两者都将其集成在他们的调试模块上。

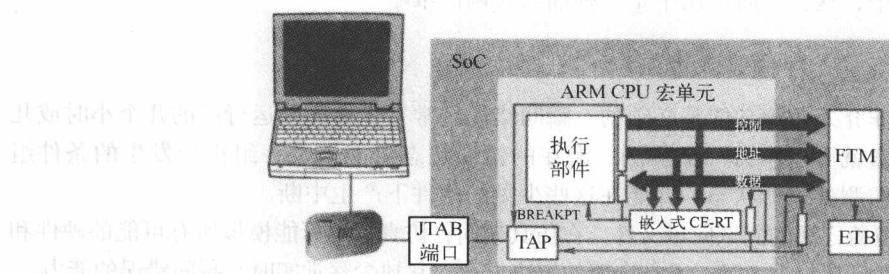


图 6-14 ARM JTAG TAP 接口（经 ARM 有限公司许可使用）

## 6.7 ARM SoC 调试

有两种方法可以完成 ARM 的片上调试。EmbeddedICE 功能模块监视处理器的地址总线、数据总线和控制总线，可以监视在处理器与其他任何逻辑之间的连接，这就意味着能够确定任意时刻的执行单元的状态。

对 IDE 来说，EmbeddedICE 功能模块能够装载带有特定数据值的硬件地址。当断点条件匹配时，将会产生中断，终止处理器执行，控制权转交给 IDE，并做进一步的分析。

在实时环境中，我们想要能够跟踪那些导致产生断点条件的指令，这样便能够重现程序执行期间到底发生了什么。更为重要的是指令、数据和控制追踪都应当是非介入性的，不能对原系统有影响。图 6-15 显示的结构中增加了跟踪缓冲存储器。

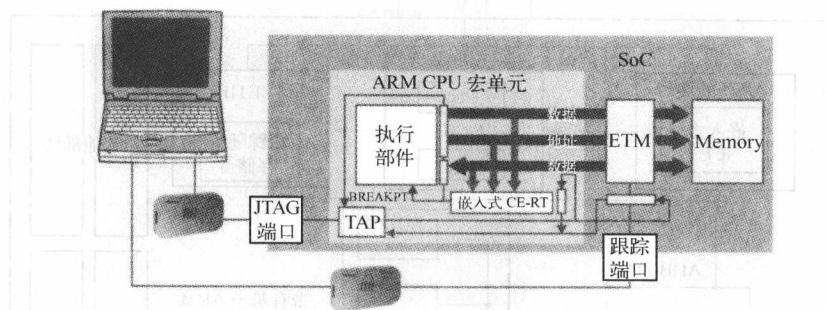


图 6-15 跟踪缓冲存储器（经 ARM 有限公司许可使用）

跟踪缓冲存储器的容量能够达到 256K 个字，甚至更多。对于一个瞬间发生的事件，程序可能运行了几个小时或几天，才会等到一个断点的产生。断点产生时，处理器获得的信息越多，对于分析越有利。处理器每秒执行几百万条指令，即使一个 512K 字的跟踪缓冲存储器，也仅能执行几毫秒的实时指令。

指令在跟踪缓冲存储器中是以十六进制代码编译格式存储的。IDE 工具套件提供了一种能够将指令序列重新组建成原始形式的方法，这样就可以进行代码级的调试，这也是 IDE 功能强大的表现。

目前，对于具有多核处理器的 SoC 的需求越来越多，如图 6-16 所示，多核 SoC 特别适合于高性能的网络交换和路由器应用。同样，基本的 JTAG TAP 也可以扩展成多核。当然，这还存在着一定的限制因素，但其必将为先进的 SoC 多核设计引入调试手段。

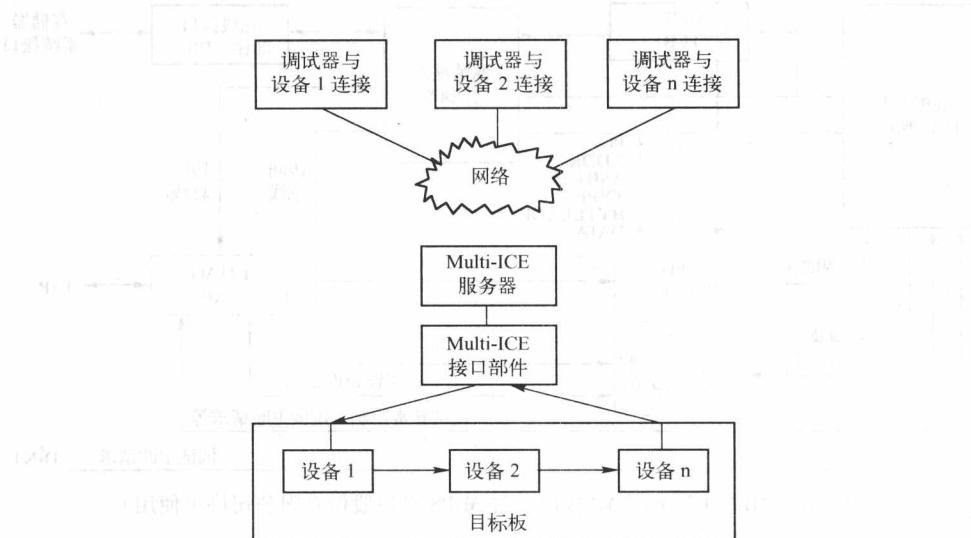


图 6-16 ARM 多核调试（经 ARM 有限公司许可使用）

## 6.8 MIPS SoC 调试

MIPS 与其合作伙伴一起开发出了 JTAG 的一种扩展版本，称为 EJTAG，用于系统调试。它定义了三种基本的组件，即测试存取口（TAP）、调试控制寄存器和硬件断点单元，通过在处理器内核基础上增加了少量的逻辑来实现，如图 6-17 所示。

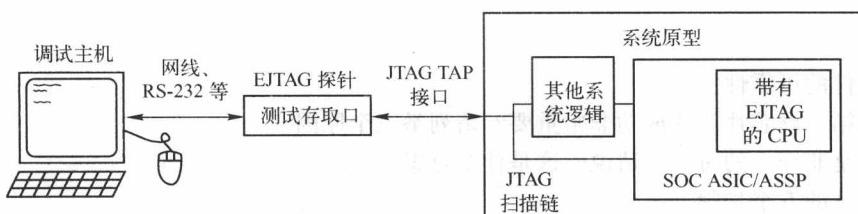


图 6-17 MIPS EJTAG 调试系统（经 MIPS 科技股份有限公司许可使用）

### EJTAG 功能

由 MIPS 公司定义的 EJTAG 具有如下五个基本职能：

- 板外 EJTAG 内存；
- 硬件断点；
- 单步执行；

- 通过 EJTAG TAP 访问系统；
- 调试断点指令。

MIPS 更加注重增加能够提高追踪能力的资源，因此定义了一个扩展的 JTAG 物理接口，如图 6-18 所示。该接口能使跟踪信息以并行方式输出，而不是串行。在处理大容量追踪缓冲区时，该项技术能够显著提高调试性能。

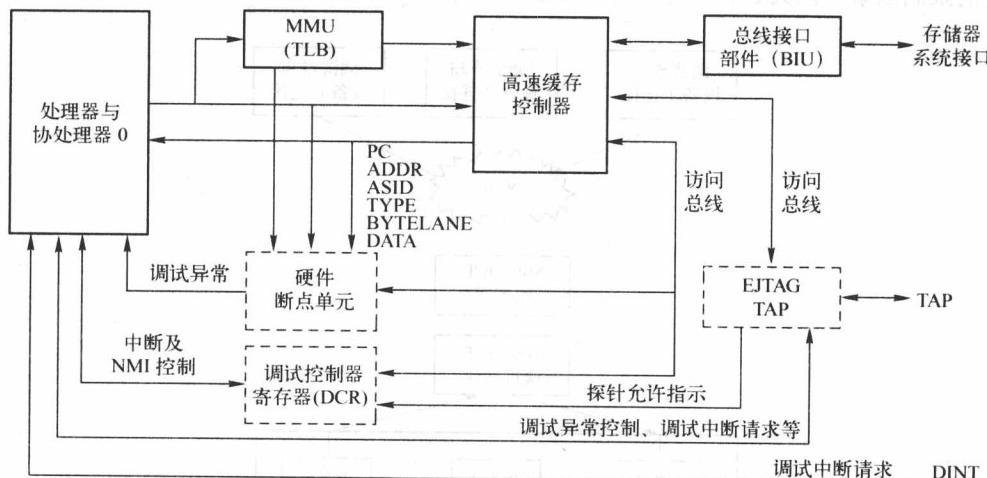


图 6-18 MIPS EJTAG TAP 接口 (经 MIPS 科技股份有限公司许可使用)

在使用任何设计工具时，都要权衡一些利弊因素。设计调试是一项复杂的工作，商用 COTS 微控制器的设计方法已经被用来开发和调试了将近 30 年。该方法已经使用很久，但却是产品从设计初期到生产销售的必经之路。

SoC 设计的生产工艺不断发展，虽然微处理器内核嵌入在众多的门级逻辑中，也能为其提供调试能力，但这是一项艰巨的任务。MIPS 和 ARM 公司都已经开发出了相应的调试策略，以应对这一挑战。两家公司都采用 JTAG TAP 方案，并将其整合到了稳定的 IDE 环境中。嵌入式微控制器调试战略将继续演变，并与 SoC 设计技术一同发展。

## 习题

1. 什么是软件的可靠性？
2. 为什么说第一款芯片流片成功非常重要？请列举三个原因。
3. 当说探针是非介入的时候，请说明这是什么意思。
4. 请列举调试的五个步骤。
5. 为什么要使用软件编辑器。
6. 仿真器的主要目的是什么？
7. 瞬时错误是什么意思？
8. 为什么要使用跟踪缓冲存储器？
9. 断点是如何使用的？
10. 源代码级调试是什么意思？
11. 请列出 EJTAG 的五种基本功能。

## 参考文献

- ARM Embedded Trace Macrocell, ETMv3.4 architectural specification.* 2004–2007 ARM Limited.
- ARM Multi-ICE user guide.* Version 2.2. 2004–2007. ARM Limited.
- Development tools data sheet.* 2008. Microchip Technology.
- Dienstbeck, R. March 2006. *Tracking the virtual world.* White Paper. RTOS Integrations, Lauterbach, GmbH.
- EJTAG specification.* Document Number MD00047 Revision 3.10. MIPS Technologies, July 2005.
- Green Hills Probe data sheet.* 2008. Green Hills Software, Inc.
- Green Hills Probe user's guide.* 2008. Green Hills Software, Inc.
- Microprocessor debug interface (MDI) specification.* Document Number MD00412, Revision 02.12. MIPS Technologies, July 2005.
- MPLAB Intergrated Development Environment (IDE) brochure, April 2008.
- Orme, W. *ARM Embedded Trace Macrocells presentation.* ARM Limited.
- PDtrace interface specification.* Document Number MD00136, Revision 3.01. MIPS Technologies, May 2003.
- 16-bit embedded control developer's resource. February 2008.

## 第7章 |

Embedded Microcontrollers and Processor Design

编文

# 串行数据通信

- 章节目标：介绍在嵌入式控制器应用中的主要串口通信协议

- 学习内容：

1. 工业标准 UART 控制器的功能
2. 应用于汽车通信中的 CAN/LIN 协议介绍
3. 多芯片系统设计中的 I<sup>2</sup>C 和 SPI 协议
4. I<sup>S</sup> 协议音频应用介绍
5. 通用外设接口 USB
6. 蓝牙

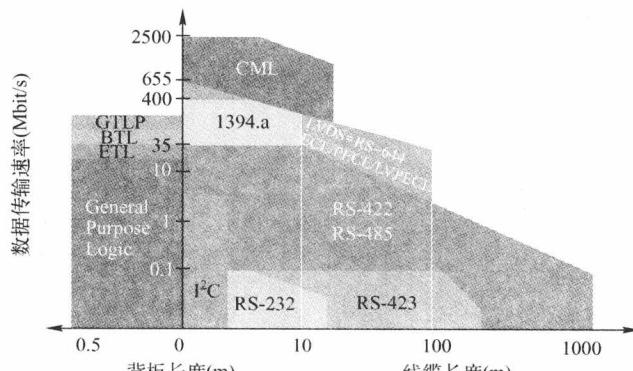
## 7.0 串行数据通信

对嵌入式控制器设计而言，串行数据通信是极其重要的。嵌入式控制器不像微处理器那样有 900 多个引脚，封装尺寸和引脚个数对产品设计起关键作用，采用最少引脚和器件实现与系统其他部件通信的能力，成为维持系统低成本的关键因素。

图 7-1a 对比了基本的串行通信总线，列出了基于微控制器设计的不同串行通信总线的优缺点

UART	CAN	USB	SPI	I <sup>2</sup> C
<ul style="list-style-type: none"> <li>● 认知度高</li> <li>● 经济有效</li> <li>● 简单</li> </ul>	<ul style="list-style-type: none"> <li>● 安全</li> <li>● 快速</li> <li>● 即插即用硬件</li> <li>● 简单</li> <li>● 低成本</li> </ul>	<ul style="list-style-type: none"> <li>● 快速</li> <li>● 即插即用硬件</li> <li>● 低成本</li> <li>● 大型系列</li> </ul>	<ul style="list-style-type: none"> <li>● 快速</li> <li>● 广泛接受</li> <li>● 低成本</li> <li>● 大型系列</li> </ul>	<ul style="list-style-type: none"> <li>● 简单</li> <li>● 认知度高</li> <li>● 广泛接受</li> <li>● 即插即用</li> <li>● 大型系列</li> <li>● 经济有效</li> </ul>
<ul style="list-style-type: none"> <li>● 功能有限</li> <li>● 点对点</li> </ul>	<ul style="list-style-type: none"> <li>● 复杂</li> <li>● 源自汽车</li> <li>● 有限系列</li> <li>● 固件昂贵</li> </ul>	<ul style="list-style-type: none"> <li>● 需要强大主机</li> <li>● 无即插即用软件，需指定驱动</li> </ul>	<ul style="list-style-type: none"> <li>● 无即插即用硬件</li> <li>● 没有固定标准</li> </ul>	<ul style="list-style-type: none"> <li>● 有限速率</li> </ul>

a) 不同串行总线的优缺点



b) 串行总线数据速率 (经 NXP 半导体公司许可使用)

图 7-1

点。在大批量设计中，最好选用成本最低且简单易用的串行通信总线。图 7-1b 提供了不同串行通信总线的传输距离和传输速率的对比。

## 7.1 UART

UART (universal asynchronous receiver transmitter, 通用异步收发器) 是目前使用的最基本的串行接口类型，发送和接收串行数据简单，通常用于调制解调器和 PC 机之间的通信，也常用作基于计算机设备的直接连接，如 Internet 路由器。Freescale 微控制器通过串行通信接口或 SCI 实现 UART 功能。

通用同步/异步收发器 (USART) 具有同步模式，但其同步模式通常很少使用，计算机与计算机之间的高速串行数据传输已被通用以太网取代。

### 7.1.1 异步模式

因为在异步模式中没有时钟信号，因此接收器必须要建立一条能实现同步数据传送功能的线路，主要通过固定位速率 (bit/s) 以及起始 (START) 位和结束 (STOP) 位来实现。发送和接收双方都必须设为相同的异步模式和传输速率，如图 7-2 所示。



图 7-2 UART 帧格式

通用 UART 通常都有多种数据字格式的选择。National Semiconductor 公司推出的 16550 即是一款具有工业标准功能的 UART，其可选格式有：

- 5 ~ 8 位数据位；
- 1 位或 2 位停止位；
- 奇/偶校验位。

图 7-2 所给出的信号中，信号在起始位保持低电平，Tx/Rx 信号通常处于高电平。数据传输首先从最低有效位 (least significant bit, LSB) 开始，接着传送 5 ~ 8 位数据位，在最后一位数据位传送之后，紧接着传送停止位，停止位通常是逻辑 1。数据传送结束时，Tx 引脚保持高电平。结束位传输完成以后，下一次传输的起始位又会出现，如图 7-2 所示。

图 7-2 中的波形代表微控制器的 Tx 或 Rx 引脚上的信号，起始位是逻辑 0，结束位是逻辑 1。必须要注意，数据是从 LSB 开始传递的，所以数据是逆序的二进制格式。尽管 RS - 232 使用负电平逻辑 1，但传送数据并未倒置。通常，使用 UART 实现 RS - 232 通信功能时，信号必须通过收发芯片进行倒置和电平转换。

例如，Microchip PIC 系列微控制器实现了 9 位数据模式，当使用奇偶校验或需要额外的停止位时，该模式十分有用。要实现奇偶校验，应根据采用奇校验还是偶校验，将第 9 位置 1 或清 0，以使总的数据位满足奇校验或偶校验。如果使用两位停止位，则将第 9 位置 1，使信号在第 8 位数据位后保持两位高电平。

奇偶校验位是一位附加位，通过该位置 1 或清 0，使传送的逻辑 1 的个数为偶数或奇数。如果使用奇偶校验，并希望至少传送 1 位，最好使用奇校验。图 7-3a 和图 7-3b 给出了奇偶校验的实例。

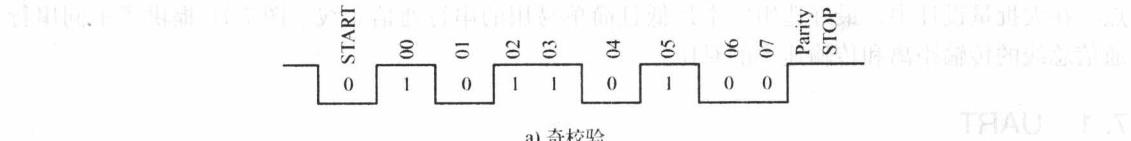


图 7-3

如今，信号集成度很高，由显示终端或调制解调器传送数据十分可靠，通常不再使用奇偶校验。例如，为了使用超级终端从一台 Windows 计算机连接到 CISCO 路由器串口，可能需要使用 8 N 1 模式传输，即 8 位数据位、无校验、1 位停止位，所以每一字符总共需传输 10 位。

### 7.1.2 发送/接收缓冲器

为了发送数据位，并行的二进制字需要装载到移位寄存器，然后按照建立起来的位速率，由时钟指挥依次移出寄存器。图 7-4 给出了 Microchip 公司的并串转换移位寄存器，它被加载在保持寄存器，数据被装入一个叫做 TXREG 的暂存寄存器中。

UART 用串并移位寄存器接收串行数据流，因而，只需使用有限的时间去读取移位寄存器中的数据。在这段时间内，一个新的数据位可能在由 RX 引脚产生，为避免时序问题，应在串并转换寄存器与存储器间设置一个缓冲器，如图 7-5 所示。

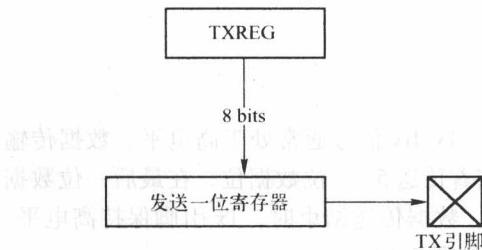


图 7-4 UART 发送缓冲器

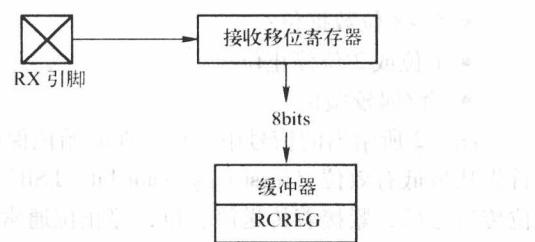


图 7-5 UART 接收缓冲器

NSC 16650D UART 使用一个 16 字节长的 FIFO（先入先出）缓冲器来完成发送和接收，可以通过减少微控制器的中断次数来增加传输速率，提高可靠性。图 7-6 给出了国家半导体 16550D FIFO 的实现。

UART 可以报告的错误包括溢出错误、帧错误及校验错误。使用 FIFO 可以帮助消除溢出错误。溢出错误是指数据传送太快，输入数据字节还未处理完，就被下一个传送数据冲掉的错误。

当接收数据中的有效位数与指定的奇校验或偶校验不一致时，便会产生校验错误。最好使用奇校验最好，如果采用奇校验，至少会在传输帧中强制数据翻转一次。

最后，当起始位或停止位不能正确识别时会出现帧错误。此时，由于停止位识别问题，接收器将不能正确地接收数据。

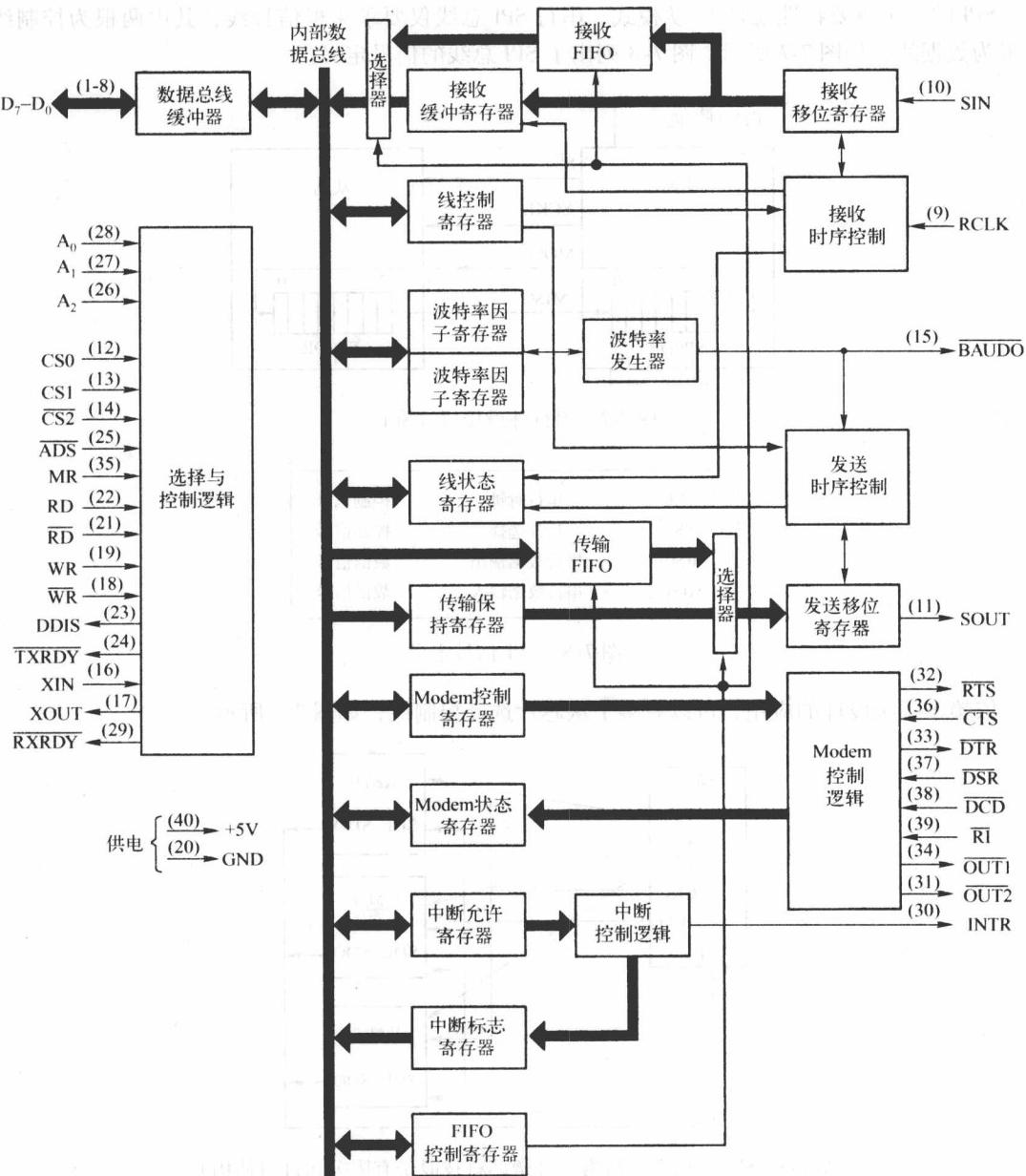


图 7-6 NSC 16550D 结构框图 (经 NSC 许可使用)

## 7.2 串行外围接口 SPI

在芯片内部采用串行通信正变得越来越流行。串行外围接口（Serial Peripheral Interface，SPI）仅仅只是其中的一个，该协议由 Freescale 命名，其他一些主要的串行协议有 I<sup>2</sup>C、CAN/LIN、USB 和 I<sup>2</sup>S。这些将在本章后面几节进行介绍。

SPI 是一种同步串行接口，可传输 8 位字节格式数据，同一时刻只能传输 1 位。SPI 可以将外部设备与另一台带有 SPI 接口的微控制器相连，这为多微控制系统设计提供了可能。

SPI 的一个重要特性就是主/从模式，串行 SPI 总线仅要求 4 根信号线，其中两根为控制线，两根为数据线，如图 7-7 所示，图 7-8 列出了 SPI 总线的信号定义。

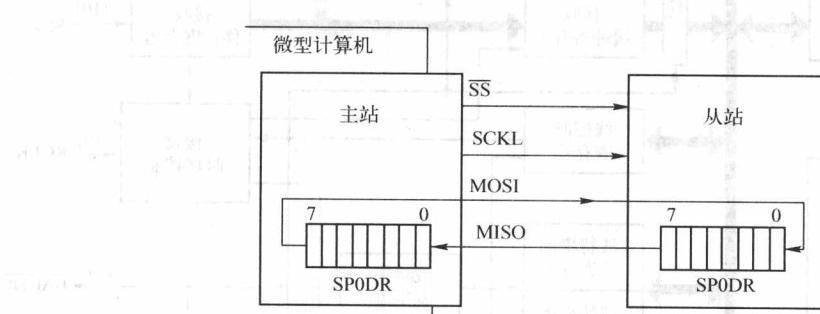


图 7-7 串行外设接口 (SPI)

SCLK	串行时钟	控制信号
SS	从片选择	控制信号
MISO	串行数据输出	数据信号
MISI	串行数据输入	数据信号

图 7-8 SPI 信号定义

依赖于系统设计的应用，可以有多个从芯片选择控制线，如图 7-9 所示。

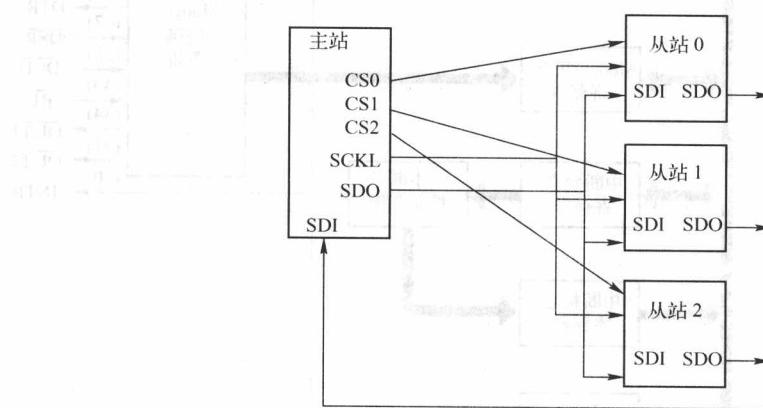


图 7-9 SPI 主从系统结构 (经微芯科技股份有限公司许可使用)

图 7-10 给出了 Microchip PIC18F45xx 系列 SPI 接口的实现。目前，还没有公共领域规范对 SPI 协议进行定义，也没有 IEEE 的认定。然而，Freescale 设计的 SPI 规范已经被广泛采用，可看作事实上的标准。如果注意 PIC 引脚定义，会发现其定义了 SDO 和 SDI 信号，而不是 MISO 和 MISI 信号，然而，其功能却完全相同。

如图 7-11 给出了 SPI 信号之间的时序。由本例可以发现，LSB 位先被移出，8 个时钟周期后，一个完整的字会被发送或接收完成，SS 信号（或从片选择）会返回正常的高电平状态。

SPI 数据传输速率可达几 Mbit/s，通常只受限于主/从 SPI 接口信号规范。该接口的主要问题是地址容量不足，每一从芯片都需要一根独立的从芯片选择线。对于简单的主/从设计来说，SPI 是非常不错的选择。

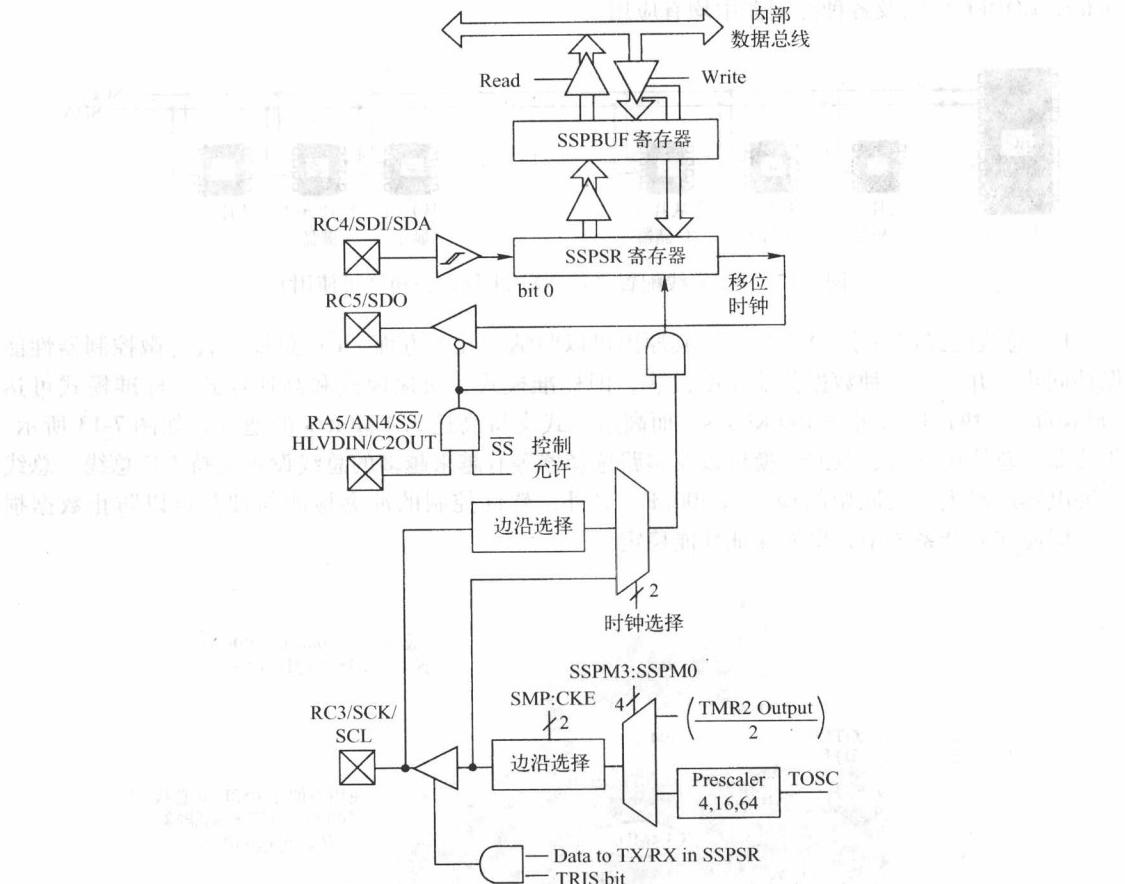


图 7-10 Microchip PIC 18F45xx 系列 SPI 实现（经微芯科技股份有限公司许可使用）

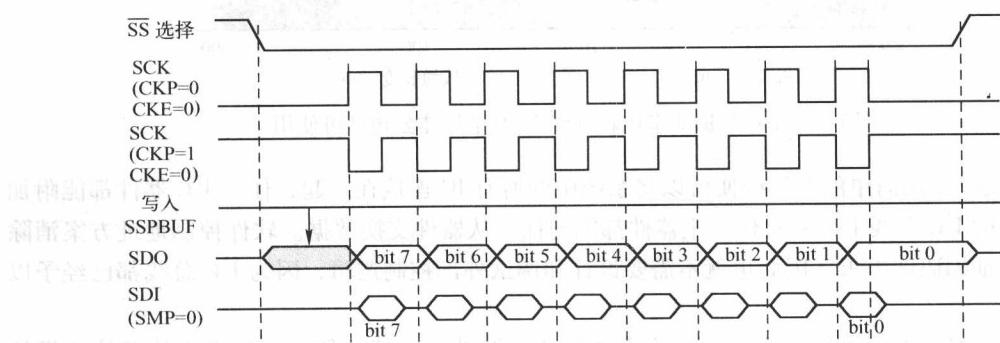


图 7-11 SPI 时序图（经 NXP 半导体公司许可使用）

### 7.3 I<sup>2</sup>C 总线

I<sup>2</sup>C 总线 (Inter-Integrated Circuit, I<sup>2</sup>C) 是一种控制总线 (见图 7-12)，为系统内部集成电路之间提供通信连接。该总线早在 20 世纪 80 年代由 Philips 公司推出，仅包含简单的两根线以及由软件定义的协议，现在已演变成系统控制的国际标准。该协议几乎可以应用在控制系统的方方面面，从温度传感器和电压变送器到 EEPROM、GPIO、A/D 和 D/A 转换器、编解码 (coder-de-

coder, CODEC) 以及各种控制器中均有应用。

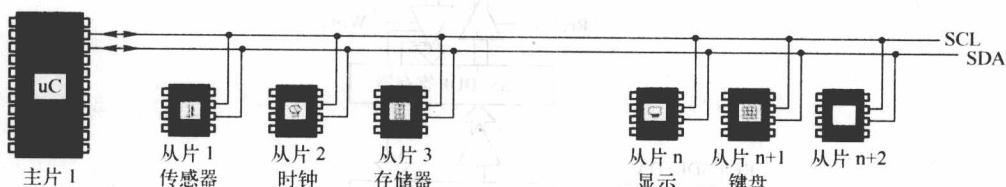


图 7-12 I<sup>2</sup>C 总线配置 (经 NXP 半导体公司许可使用)

I<sup>2</sup>C 总线已经流行了 20 多年，究其原因可以归纳为几个方面。I<sup>2</sup>C 总线一直与微控制器性能保持同步，并且有三种数据传输速度模式，即标准模式、快速模式和高速模式。标准模式可达 100 Kbit/s，快速模式可达 400 Kbit/s，而高速模式支持高达 3.4 Mbit/s 的速度，如图 7-13 所示。集线器、总线中继器、双向交换机以及多路选择器等有越来越多的总线设备支持 I<sup>2</sup>C 总线，总线扩展电容已经大于其原始的最大值 400 pF。另外，软件控制的冲突检测与仲裁可以防止数据损坏，即使在复杂系统中，也能保证性能稳定。

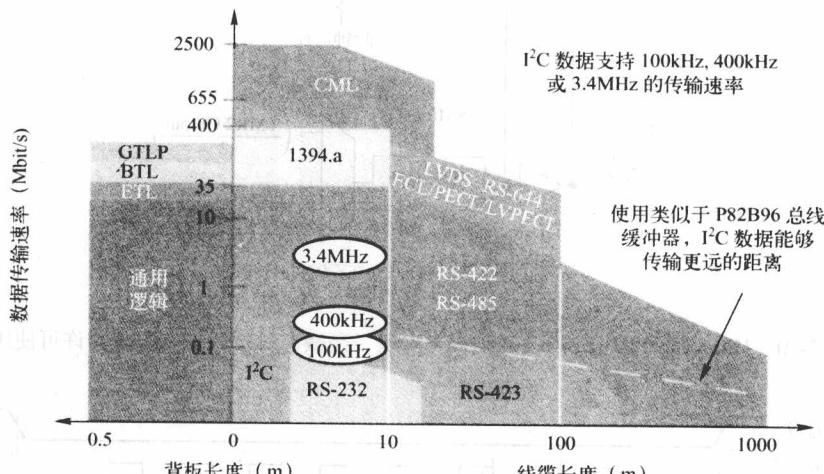


图 7-13 I<sup>2</sup>C 数据速率比较 (经 NXP 半导体公司许可使用)

I<sup>2</sup>C 总线仅使用简单的两根线便可以将系统中的所有 IC 连接在一起，任一 I<sup>2</sup>C 器件都能附加在一条通用的 I<sup>2</sup>C 总线上，并且任一主器件都能与任一从器件交换数据。软件控制地址方案消除了对硬件地址编码的需求，因此也就不需要设计和调试外部控制逻辑，因为 I<sup>2</sup>C 总线都已经予以提供。

通过在已经存在的总线上简单地裁剪或增加新的器件与功能，便可以根据方块图快速设计出最终的硬件。I<sup>2</sup>C 总线还可节约空间，节约成本；两线结构意味着更少连线，所以 PCB 板能更小；由于互联线减少以及需要验证的信息源减少，测试与调试也变得更加容易。经过几代系统的演变，I<sup>2</sup>C 设备变得很容易进行添加或删除，而不影响系统的其他部件。

### 7.3.1 I<sup>2</sup>C 总线如何工作

任何 I<sup>2</sup>C 设备都可添加到 I<sup>2</sup>C 总线上，并且每一设备都能与总线上任一主设备对话，来回传递信息。总线上至少需要一个主控制设备（如微控制器或 DSP），但实际上可能有多个主设

备，所有主控制设备有相同的优先权，能很容易地在 $I^2C$ 总线上进行添加或删除。总的总线电容要求低于400pF，据此可以算出20~30个标准设备或10m的信号线。对于逻辑低电平需要满足3mA的驱动能力，对于上拉电阻为 $2\sim10\text{k}\Omega$ 的开漏极总线需要提供0.4mA的驱动能力，如图7-14所示。

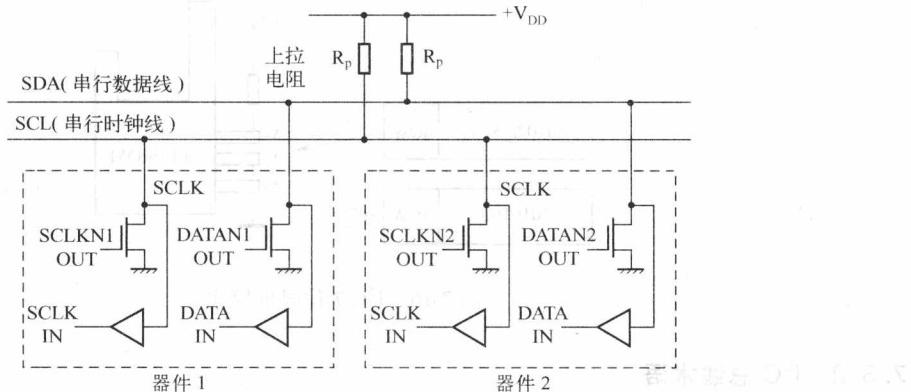
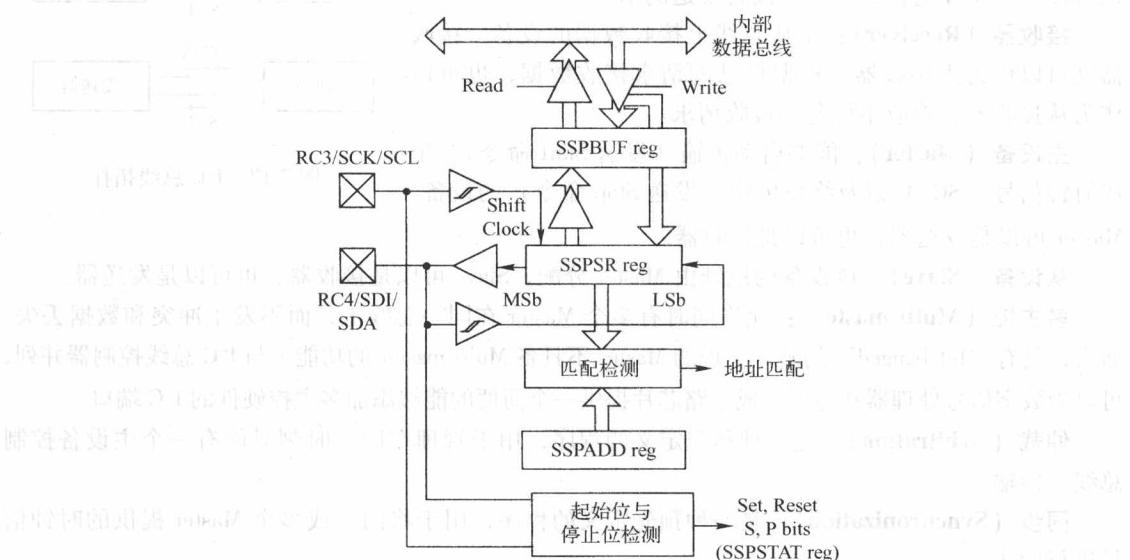
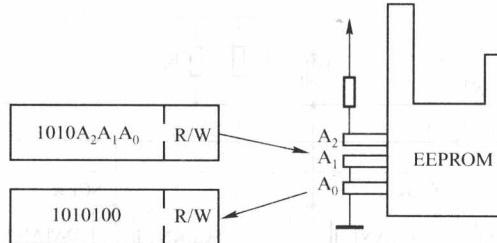
图 7-14  $I^2C$  总线信号示例

图7-15给出了 $I^2C$ 总线的一个典型微控制器(Microchip PIC18F4520)的实现功能框图，此处用作内部设备。

图 7-15 Microchip PIC18F4520  $I^2C$  功能框图(经微芯科技股份有限公司许可使用)

在标准配置中，每个设备都有唯一的7位地址，如图7-16所示，从而使主控制设备明确地知道在和谁通信。通常，高四位固定，用于指定设备种类（如1010分配给了串行EEPROM）。低3位(A2、A1、A0)通过地址引脚可以实现编程，允许最多8个不同的 $I^2C$ 地址组合，因此，最多允许相同种类的8个设备在同一根 $I^2C$ 总线上工作。这些引脚中，如果是逻辑1，则连接到VCC；如果是逻辑0，则连接在GND。7位编址理论上允许至多128个设备连接在同一根 $I^2C$ 总线上，但这些地址中有一些是保留的，主要用于特殊命令，所以实际上连接设备数目限制在120个左右。

**扩展地址模式**同样也支持10位地址，允许连接最多1024个设备。它并不改变在I<sup>2</sup>C总线规范中定义的地址格式，允许使用已存在规范中的保留地址。10位编址并不影响已经存在的7位地址，它允许7位或10位设备连接到相同的I<sup>2</sup>C总线上，并且这两种类型的设备可使用在标准、快速、高速模式系统中。

图 7-16 I<sup>2</sup>C 7 位寻址模式

### 7.3.2 I<sup>2</sup>C 总线术语

如图 7-17 所示为 I<sup>2</sup>C 总线拓扑结构方块图示意图，以下内容定义了 I<sup>2</sup>C 总线的基本术语。

**发送器 (Transmitter)**：用于发送数据到总线上的设备。

发送器既可以作为主发送器，按自己要求将数据发到总线上，也可以作为从发送器，响应主设备发送的请求。

**接收器 (Receiver)**：是从总线上接收数据的设备。接收器既可以作为主接收器，根据自己的请求接收数据，也可以作为从接收器，响应主设备的接收请求。

**主设备 (Master)**：能够启动传输（发送 Start 命令），生成时钟信号（SCL）以及终止传输（发送 Stop 命令）的设备。

Master 可以是发送器，也可以是接收器。

**从设备 (Slave)**：该设备的地址由 Master 分配，Slave 可以是接收器，也可以是发送器。

**多主控 (Multi master)**：允许同时有多个 Master 在同一总线上，而不发生冲突和数据丢失。通常，具有“bit-banged”的软件实现的 Master 不具备 Multi master 的功能。与 I<sup>2</sup>C 总线控制器并列，可以为数字信号处理器和专用集成电路芯片提供一个简便的能够添加多主控硬件的 I<sup>2</sup>C 端口。

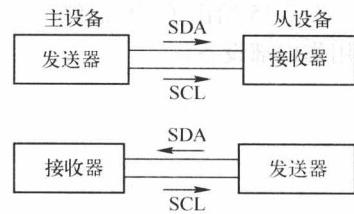
**仲裁 (Arbitration)**：是一种预先定义的程序，用于管理在同一时刻只能有一个主设备控制总线的传输。

**同步 (Synchronization)**：是一种预先定义的程序，用于将两个或多个 Master 提供的时钟信号进行同步。

**串行数据信号 (Serial Data Signal)**：数据信号线（SDA）

**串行时钟信号 (Serial Clock Signal)**：时钟信号线（SCL）

I<sup>2</sup>C 总线的目标设备地址在第一个字节发送，该起始字节的最低有效位表明主设备是将要发送（写入）数据，还是从接收器（从设备）接收（读取）数据。每个传送数据的序列都必须以 Start 状态开始，以 Stop 或 ReStart 结束。如果同一 I<sup>2</sup>C 总线有两个 Master，当二者在同一时刻发出 Start 命令来控制总线传输时，便需要仲裁程序进行裁断。一旦一个 Master（如微控制器）接管了总线，那么其他的 Master 将处于闲置状态，它们必须等到第一个 Master 发送了停止状态，并将总线置于闲置状态后，才能控制总线。

图 7-17 I<sup>2</sup>C 总线拓扑

### 7.3.3 总线传输术语

图 7-18 给出了总线传输的基本时序示意图，有关总线传输的基本时序和术语如下所述。

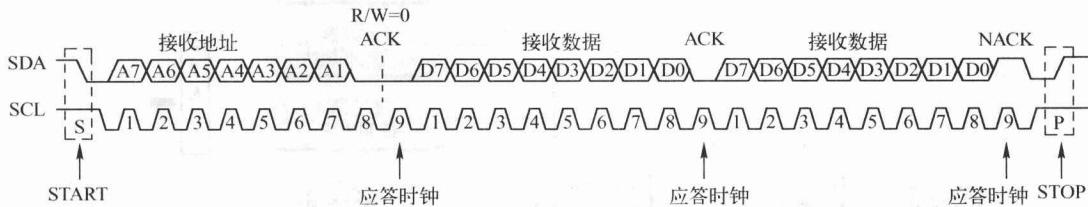


图 7-18 I<sup>2</sup>C 总线传输

**F (FREE):** 总线闲置或空闲，数据线 SDA 和 SCL 时钟都处在高电平状态。

**S (START) 或 R (RESTART):** 数据传送以 Start 状态开始。SDA 数据线电平由高变为低，而 SCL 时钟线依然保持高电平。当这一过程开始后，总线便处于忙状态。

**C (CHANGE):** SCL 时钟线变为低电平后，要传送的数据位便可以由发送器送到 SDA 数据线上。这期间，SDA 可以改变状态，直到 SCL 线不再为低。

**D (DATA):** SCL 时钟线为高电平时，SDA 线的高低位信息。在时钟高电平期间，它的电平必须保持稳定，以避免被当作起始或停止状态而错误中断。

**P (STOP):** Stop 状态可以终止数据传送。SCL 时钟线处于高电平时，如果 SDA 数据线从低电平转换到高电平，则会产生 Stop 状态。当数据传送被终止时，总线又变成空闲状态。

## 7.4 CAN 总线

现在，汽车上集成了数百个集成电路、传感器以及许多其他电子部件，许多电路与汽车功能部件之间需要进行通信，例如，当司机按下仪表盘上的前灯开关，前灯就会反应，这时就需要在仪表盘开关与汽车前灯之间进行通信。

在当前的汽车系统中，该类型的通信是通过点对点的连线完成的。如果在一个配置齐全的汽车上，将所有可能的开关、传感器、马达以及电子器件都计算在内，则需要无数的连接及专用线路，然而，网络却为复杂的车载通信提供了更加有效的方法。

车载网络 (In-vehicle networking, IVN)，也称作多路传输，是通过串行数据总线在分布式电子模块之间进行数据传输的方法。如果没有串行网络，内部模块通信需要专用的、点对点的连线，将导致连线数量繁多，价格昂贵，十分复杂，电线束也难于安装。应用串行数据总线，通过时分复用技术，可以把信号合并到一根单一的线号线上，将会减少很多信号线。并且信息会传递到一个独立的控制模块来控制每一项功能，如防抱死制动、转向灯、显示仪表等，如图 7-19 所示。

车载网络具有很多系统层次的优点，其中很多才开始被人们意识到。每一功能都需要这一数量降低的专用线路，因此也就减少了线束尺寸。系统成本、重量、可靠性、适用性以及安装便利性上都有改善。常用传感器数据，如车速、引擎温度等，在网络上都是可用的，并且其测量数据在网上共享，可以减少传感器的数量。

由于功能模块可以通过改变软件来增加，网络给汽车功能带来了更大的灵活性。现有系统需要为每一个新增功能增加额外的模块或引脚。图 7-20 列出了 CAN 总线的输入/输出特性。

**阅读** 当把这一复杂的网络应用于批量生产经济型汽车时，系统增加的成本要低于总的获益。标准的协议能够实现这一应用，汽车制造商和各种汽车工业标准化组织已经致力于研究车载网络很多年，另外，CAN、LIN 和 SAE J1850 已成为当今的主流标准。

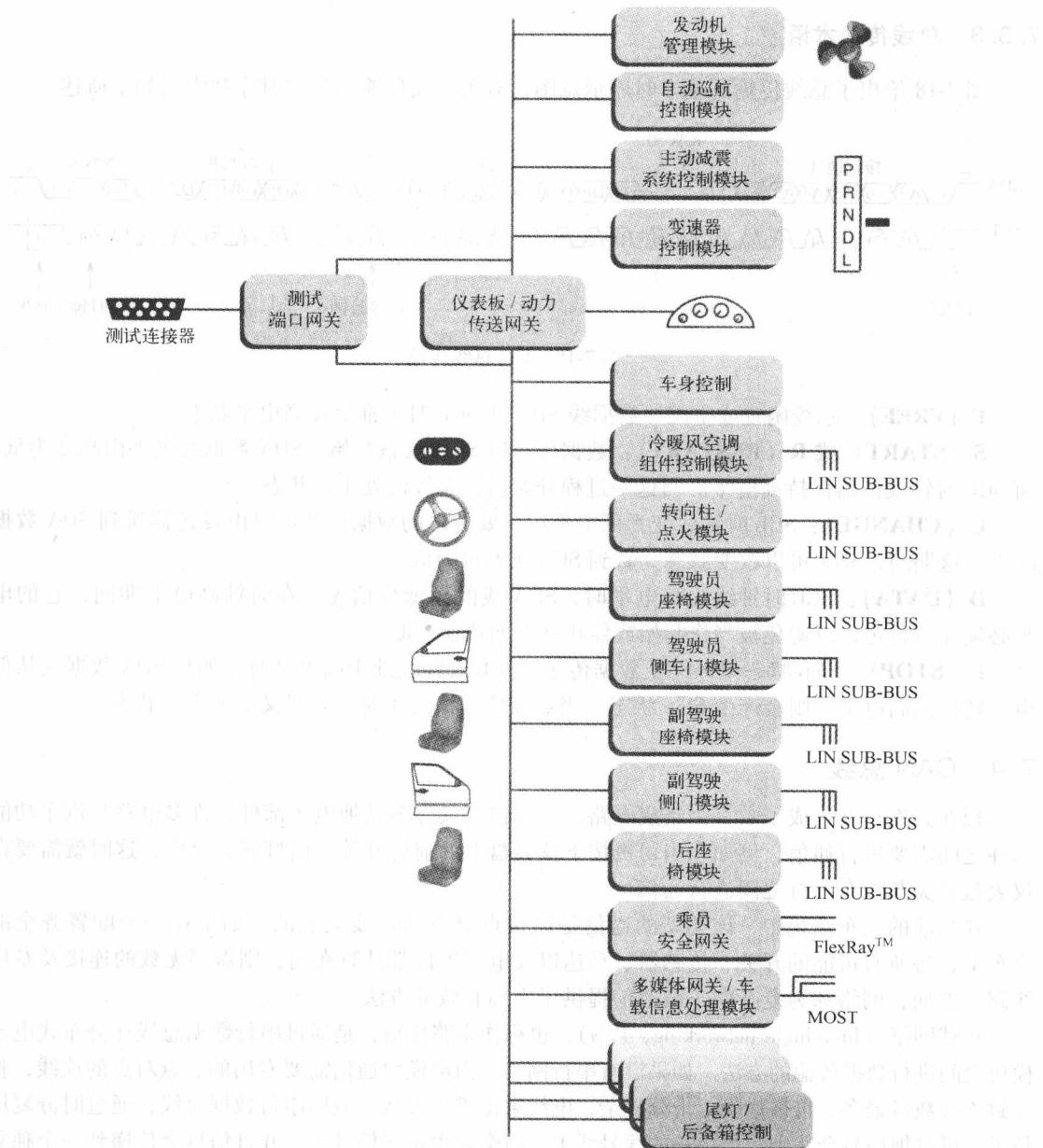


图 7-19 CAN 总线功能 (Freescale 2007 版权许可)

SAE J1850 和 CAN 2.0 采用的是载波监听多路访问/冲突解决 (carrier sense, multiple access with collision resolution, CSMA/CR) 仲裁协议，有一个多主结构，串行总线可以传递进行优先级的信息。当两个或多个主设备试图同时发送信息时，仲裁协议便会处理这些信息内容，丢弃最低优先级的数据，最高优先级的信息成功送到达目的地，而不会因为冲突破坏其内容，这样就实现了“冲突解决”。

网络的一个主要优点是具有扩展功能，并且不用增加硬件，也不会降低可靠性。例如，当网络功能成为中低档汽车的通常配置时，汽车制造商就能很容易为其提供如今只在高端汽车上才能见到的功能。

CAN 总线信号类型	数字接口最小 +4V, 最大 +5.5V
输出电压 (高)	V <sub>oh</sub> 最小 +4V, 最大 +5.5V
输出电压 (低)	V <sub>ol</sub> 最小 0V, 最大 +1.5V
输出电压	+16V (绝对最大额定值)
输出电流	100mA
电阻	+/- 接线柱之间 124Ω 终端电阻
电路类型	差分
每位数据传输时间	1Mbps 时 1μs, 0.5Mbps 时 2μs, 0.25Mbps 时 4μs
编码格式	不归零码 (NRZ)
发送 / 接收频率	40 米内时 1Mbit/s
拓扑结构	点对点
传输媒介	屏蔽双绞线 (STP), 9 针 D 型插头
访问控制	CSMA/CD, 带优先权的非破坏性逐位仲裁

图 7-20 CAN/LIN 总线特性

并不是所有的网络都是平等的。速度与低延时对于供电控制和汽车的动态性能来说十分关键，然而，驾驶室的部分功能，如窗户和仪表的供电，只需要超越人的感知速度。为了优化成本和控制数据通路，汽车中有多个网络已变得很平常。例如，图 7-21 所示的 CAN 网络运行于 500 Kbit/s 的速度，可以连接发动机、数据传输以及 ABS，而速度较慢的 CAN 网络或 J1850 网用于连接车门、仪器和其他电子设备。在多个网络之间，网关能够传递需要的诊断信息。

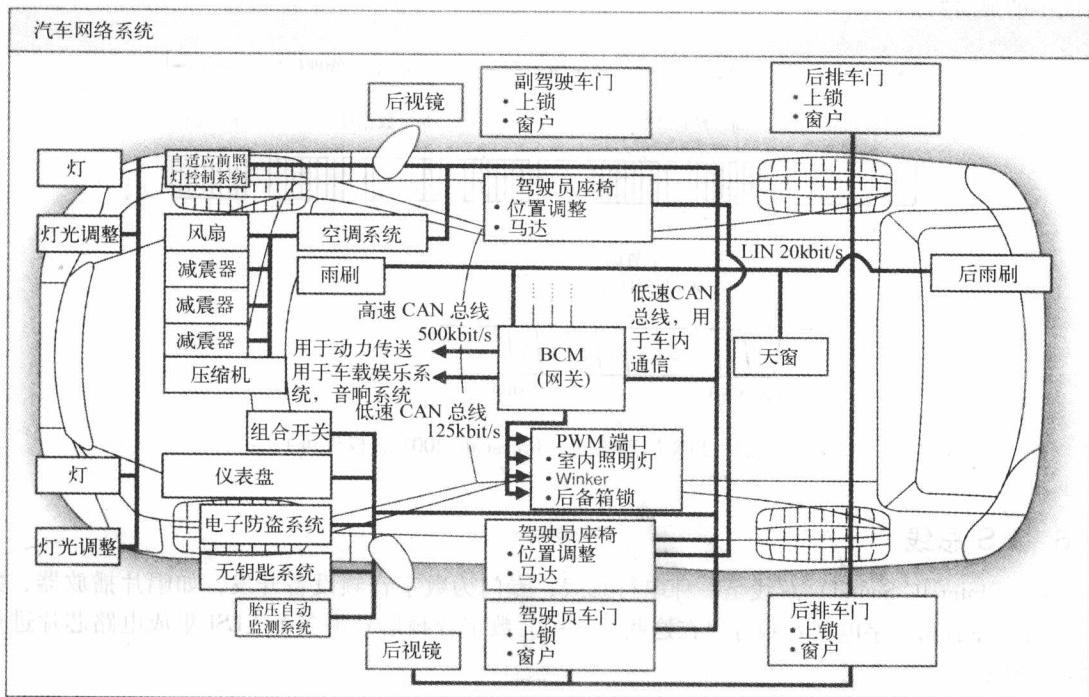


图 7-21 汽车 CAN 总线网络 (经 Renesas 科技股份有限公司许可使用)

这不仅对标准协议中的数据链路层、物理层有益处，而且对于系统设计师而言，也能感受到标准化应用层协议的优点。这些标准允许系统设计师们不用关心低层协议的详细信息，将精力

集中在应用上。然而，对于微控制器和协议设备需求的增加对该标准产生很大影响，因此，对于高效信息处理和标准协议设备的需求将会变得更加重要。

## 7.5 LIN 网络

LIN (local interconnect network，局部互连网络) 的最大特点就是节约成本。它是一个单总线网络，最大速率为 20 Kbit/s，可以限制电磁干扰 (Electro-magnetic interference, EMI)。如图 7-22 所示，LIN 结构中使用了一个专用的主设备和多个从设备，这种结构消除了仲裁协议的需求，降低了控制器的复杂性。

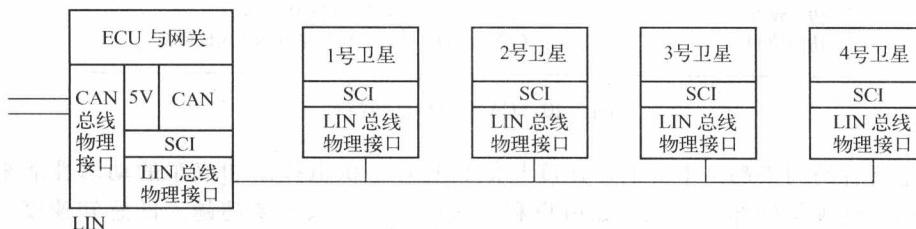


图 7-22 LIN 网络拓扑结构 (Freescale 2007 版权许可)

LIN 利用了通用的 UART/SCI 硬件接口，几乎所有的通用微处理器都支持该接口（见图 7-23），这样也会降低成本。另外，LIN 不需要从节点使用晶体或陶瓷谐振器，这是另一个节约成本的显著特点。

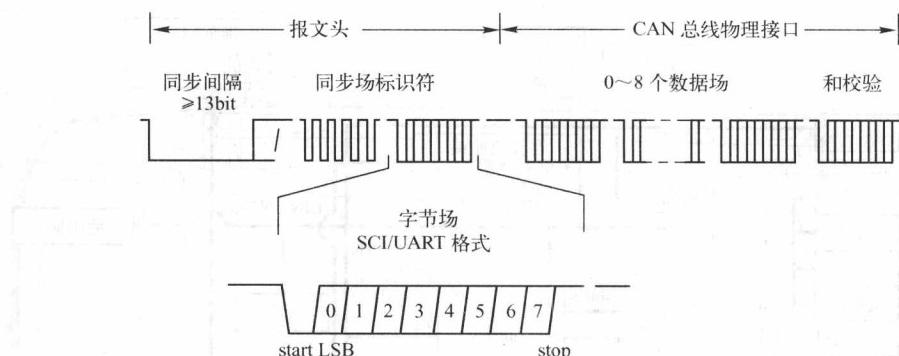


图 7-23 LIN 信息格式 (Freescale 2007 版权许可)

## 7.6 I<sup>2</sup>S 总线

I<sup>2</sup>S (Inter-IC Sound) 总线是一种串行总线，专门为数字音频设备开发，如唱片播放器、数字音频处理器和数字电视音频等。在这些系统中，数字音频信号由多个 VLSI 集成电路芯片进行处理，如：

- A/D 和 D/A 转换器；
- 数字信号处理器；
- 唱片和数据记录纠错；
- 数字滤波器；
- 数字 I/O 接口。

标准通信结构对设备和IC制造商都很重要，因为它们增加了系统的灵活性。鉴于该目的，Phillips公司开发了I<sup>2</sup>S总线，专门用于数字音频的串行连接。I<sup>2</sup>S设计中将音频信号与时钟信号分别处理。分离数据域时钟信号，就不会发生时序错误引起的抖动，进而消除对防抖动设备的需要。

I<sup>2</sup>S有3条主要信号线，一根具有时分多路数据通道的线、一根字选择线和一根时钟信号线。图7-24给出了发送器与接收器之间通过时钟(SCK)和数据线(SD)的连接示意。

这种总线只需要处理音像数据，其他信号，如译码与控制等，分别进行传输。为了使需求的引脚数量最小化，并使连线尽可能简单，使用了一个具有三根线的串行总线，包括一根用于两个时分多路数据通路的连线、一根字选择线以及一个时钟信号线，如图7-25所示。

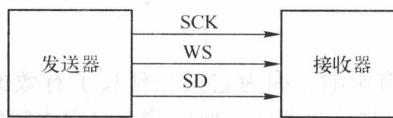


图7-24 I<sup>2</sup>S发送器用作主设备

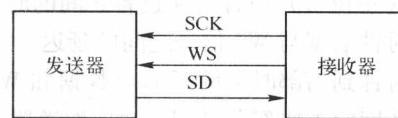


图7-25 I<sup>2</sup>S接收器用作主设备

因为发送器和接收器用于数据发送的有时钟信号相同，作为主设备的发送器必须要产生位时钟信号、字选择信号和数据信号。然而，在复杂的系统中可能有好几个发送器和接收器，很难确定谁是主设备。在这些系统中，常会有一个系统主设备控制数字音频数据在不同集成电路间传递，发送器在外部控制时钟的控制下产生数据信号，因此用作从设备。图7-26展示了一个简单的系统配置，控制器扮演了主设备的角色。需要注意的是，系统主设备可以与接收器或发送器合并，并且在软件控制下，或通过编程引脚，它可以被允许或禁止。

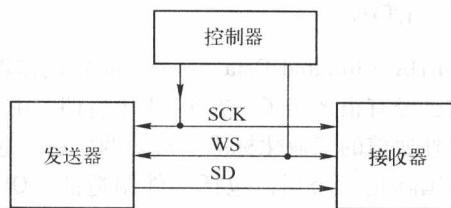


图7-26 I<sup>2</sup>S控制器用作主设备

### 7.6.1 I<sup>2</sup>S串行数据

串行数据以二进制补码的形式传递，最高有效位优先发送。首先传送MSB，是因为发送器和接收器可能具有不同的字长，发送器不需要知道接收器能处理多少位，接收器也不需要知道发送器到底传送了多少位数据。

当系统的字长大于发送器的字长时，数据要进行裁短（一些最低有效位被设置为0）。如果接收器收到的数据大于字长，最低有效位以后的数据位会被忽略。另一方面，如果接收器发送的数据不够字长位数，则其后补0。所以最高有效位具有固定位置，而最低有效位的位置决定于字长。发送器通常在字选择信号(WS)变化之后发送下一个字的最高有效位，并延续一个时钟周期的长度。

发送器发送的串行数据可能与时钟信号的上升沿或下降沿同步。然而，串行数据必须在串行时钟信号的上升沿由接收器进行锁存；并且，当发送使用上升沿同步的数据时，会有诸多限制。

### 7.6.2 I<sup>2</sup>S字选择

WS线用于传送数据通道选择：

WS=0，通道1（左）

WS=1，通道2（右）

WS 信号既可以在串行时钟信号的上升沿改变，也可以在下降沿变化，但它不必对称。在从设备中，该信号在时钟信号的上升沿被锁住。在数据信号的最高有效位发送前，WS 信号会变化一个时钟周期，这允许从设备发送器将要发送的串行数据进行同步。另外，它还能使接收器存储先前的字数据，并将输入信号清 0 来准备接收下一个字数据。

### 7.6.3 I<sup>2</sup>S 总线时序

在 I<sup>2</sup>S 的格式中，任何设备都能通过产生必要的时钟信号来扮演系统主设备的角色。从设备往往使用外部时钟输入驱动其内部时钟信号。这就意味着，如果将主设备时钟和数据与 WS 信号之间的传播延迟考虑在内，那么总延迟可以简单表示为两个延迟之和，即：

外部（主设备）时钟与从设备之间的时钟延迟；

内部时钟数据与 WS 信号之间的延迟。

外部时钟到内部时钟的延迟对数据和 WS 输入没有影响，因为它只是延长了有效的建立时间。时间裕度的主要部分是用来适应发送器的传播延迟与接收器所需要的建立时间之间的差异。

所有的时序要求都相对于时钟周期或设备允许的最小时钟周期，这就意味着将来能够使用更高的传输速率。

## 7.7 IrDA

IrDA (Infrared Data Association) 用作电视/录像机控制器、计算器、打印机、PDA 等的传输介质已经很多年了。在 1993 年后期，由 HP、IBM 和 Sharp 发起成立了一个工业组织，用于推进红外通信的工业化标准。短短两年后，这个红外数据联合会的成员就增加到 130 名，并且成员都是国际化大公司，包括组件制造商、OEM 厂商、硬件和软件公司等。

到了 1995 年，很多 IrDA 兼容产品被应用，这些产品包括笔记本电脑、掌上电脑、打印机以及用于个人计算机及打印机等的红外适配器等。与早期的红外处理器使用专用协议不同，与 IrDA 兼容的新型设备通过应用程序、制造厂家以及开发平台可以相互操作，如图 7-27 所示。IrDA 标准的主要特点有：

- 实现简单，低成本；
- 低功耗；
- 直接点对点连接；
- 数据传输高效、可靠。



图 7-27 IrDA 应用 (经 ZiLOG  
公司许可使用)

通信协议可以解决很多问题，并且通常划分为多个层次，每一层具有管理职责并为上、下层提供相应的服务。将一层层协议叠加在一起时，可得到协议栈，但这不像一叠煎饼或一摞盘子。IrDA 协议栈是协议的分层集合，尤其瞄准了点对点红外通信目标以及这方面的应用。

### IrDA 协议栈

IRDA 协议栈是一个 5 层的模型，包含 8 个基本功能，其中有一些是可选的。图 7-28 显示了它们之间的相对位置，图 7-29 的表格给出了每层的定义。

兼容通信参数自动选择和“黄页”网络信息服务等特征使得 IrDA 协议十分适合嵌入式设备。即使在消费市场，设备必须通信简单（如电视遥控器），才能被广泛接受。

有关 IrDA 的标准文件和附加信息可在 <http://www.irda.org> 网站上找到, IrDA 网站上还有相关硬件和软件供应商的查询链接。

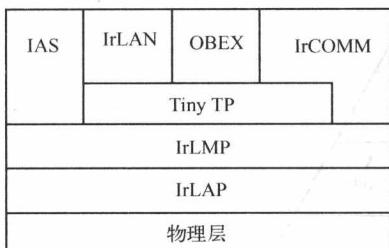


图 7-28 IrDA 堆栈结构示意

物理层	帧 / 物理层
IrLAP	链接访问协议
IrLMP	链接管理协议
IAS	信息访问服务
Tiny TP	IrDA 传输通信协议
IrLAN	局域网访问
OBEX	对象交换通信协议
IrCOMM	串行与并行端口仿真

图 7-29 IrDA 堆栈层定义

## 7.8 USB 总线

USB (Universal Serial Bus, 通用串行) 总线于 1994 年由 7 家公司发起创立, 定义为在电话和计算机之间的简单接口。就在订立设计规范的过程中, 最初的参与者制订了 USB 实现者论坛 (USB Implementers Forum, USB-IF), 现有成员已超过 1000 多名, 并从发起成员中选择成立了论坛董事, 主要包括 Agere System、Hewlett-Packard、Intel、Microsoft、NEC 和 Philips 等公司。

USB 2.0 规范支持 480 Mbit/s 的传输速率。该速率足够用于照片质量扫描仪和大容量备份硬盘驱动器, 因为 60 Mbit/s 是一个十分快速的传输速率。USB2.0 规范向下兼容 USB1.1 协议中的 1.5 Mbit/s 低速和 12 Mbit/s 全速传输速率。

USB 应用十分广泛。图 7-30 中列出了部分采用 USB 进行数据交换的产品示例。对于小型尺寸的产品, 如数码相机, 已在 USB-OTG 规范中制定了一种增强的微型 USB 接口。



图 7-30 USB 产品 (Freescale 2007 版权许可)

### 7.8.1 USB 拓扑

USB 采用分层星形拓扑结构或树形拓扑, 如图 7-31 所示。最高层是主机 (Host), 它控制树的其他部分。Host 通常与一个集线器 (Hub) 连接, 该 Hub 使总线分段, 以使多个节点 (Node)

都可以连接到总线上。每个 Hub 可以连接几个 Node 或几个 Hub。

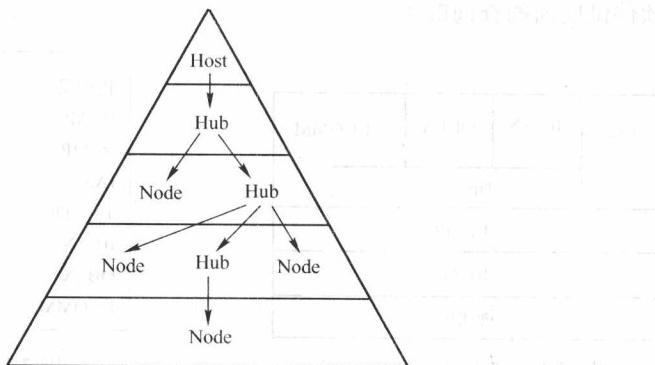


图 7-31 USB 树形拓扑

多个 Node 可以和一个 Hub 封装在一起，如带有两个 USB 接口的键盘，这种设备称为复合设备。不管连接了多少设备，Host 始终为总线主控器。一个 Hub 与另一 Hub 或 Node 之间的距离不能超过 5 m，Host 与指定节点间最多可以有 5 个 Hub。USB 总线最多支持 127 个设备连接到总线上，其中包括 Host 本身。

### 7.8.2 USB 构架

通过发送经过编码的二进制序列，USB 与连接在总线上的设备进行通信，通信速率由目标节点决定，USB 2.0 高速设备的通信速率为 480 Mbit/s。Host 是拓扑结构的根节点，是总线的控制器，也就是说，由 Host 安排和启动所有的数据传输。

USB 支持异步和同步数据传输模式。异步传输不能保证传送能在指定时间内完成，但能保证传送数据的准确性。同步传输模式能保证数据在指定时间内完成，但不能保证数据传输的准确性。

Host 和设备看待传输媒介是不同的。Host 与端点之间的数据链接称为管道（Pipe），这些管道是一束单向数据传输通路。应用程序可以通过不同的管道与功能设备进行通信。同时，设备通过端点（Endpoint）与 Host 打交道，端点是单向的数据源或数据接收者，每个端点都有确定的优先级，以决定由哪个端点与 Host 通信。

USB 的另一优点是可以通过总线提供电力。每个设备允许消耗一定数量的电能，具体数量在网络配置时确定。如果一个设备需要的电能超过了最大可用量（500 mA, 5 V），就必须自己供电。自带电源的设备使用其自身提供的电力，而由总线供电的设备使用 USB 总线提供的电力。

USB 提供设备的识别和枚举。当一个设备添加到网络中后，由主机查询某特定字符，并分配一个地址。然后，Host 会启动设备驱动，并与驱动器交流相应的信息。为了让 USB 设备支持热拔插，USB 设备驱动要能动态的装载和卸载。设备的驱动器仍然通过操作系统来交流，并通过适当的管道传送数据。

USB 数据传送使用位填充的 NRZI 编码方案，也就是说，在 USB 数据线上每 7 个位时间内，编码后的数据必须至少翻转 1 次。这种编码方式将产生足够的翻转次数，来允许设备进行时钟恢复。图 7-32 展示了整个数据通信方案。

数据包由二进制位构成，可以在 USB 总线上自动传递。数据包由同步序列、包识别码（Packet Identifier, PID）构成，包识别码紧跟在同步序列的后面，为 4 为二进制编码在其后紧跟

其反码构成，数据长度和循环冗余校验（Cycle Redundancy Check，CRC）位于 PID 的后面，CRC 的类型会根据传输类型的不同而改变。

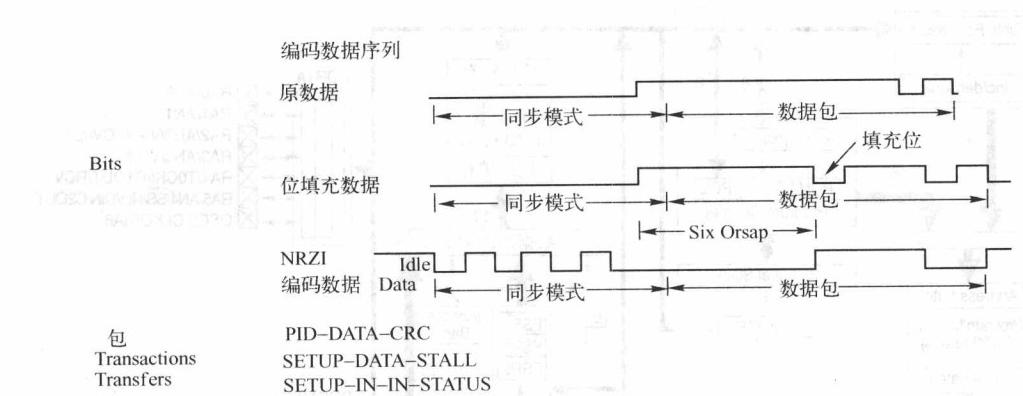


图 7-32 USB 数据通信方案

数据传输由数据包构成，可以传输命令或数据。所有的数据传输都包含一个令牌包，如果要传输数据，会紧跟一个数据包。当建立的传输管道是异步时，则会紧跟一个握手信息。

根据管道数据传输的类型，传输可以一次完成，也可能需要多次传输完成。控制传输至少需要三次传输，而同步传输只需要一次即可完成。

### 7.8.3 USB 物理连接

标准 USB2.0 电缆带有 A 型（母）和 B 型（公）连接器。对于小型尺寸的设备，USB OTG 补充规范 1.0a 到 USB 2.0 规范中，定义了 Mini-A 和 Mini-B 类型连接器。

USB 电缆采用 4 芯屏蔽电缆，有两根是电源线——+5 V（红）和地线（棕），两根双绞线（黄和蓝）用于数据信号传输。电源线可支持 5 V 500 mA，足以支持设备运行，如驱动外置 2.5 英寸的硬盘驱动器。数据线支持 USB2.0 规范中的最大数据通信速率 480 Mbit/s。

### 7.8.4 USB 接口

USB 设备支持热拔插，这意味着可随时将设备从主机或 Hub 拔出或插上。该特点使 USB 使用非常方便。例如，USB 接口的外置硬盘可在多个计算机进行备份使用。

USB 接口的另一特点是能够将设备进行菊花链接。一个 USB 键盘可以包含 USB 接口，并支持鼠标等外设。此外，该键盘可能添加到含有 USB 接口的显示器上。这样，可以利用 USB 的这一特性，减少直接连接到计算机上的连线数量。

很多嵌入式微控制器都有一个 USB 2.0 主机接口，这大大简化了在嵌入式应用中使用 USB 设备进行系统设计的难度。图 7-33 中，Microchip PIC18F4550 便内置了 USB 功能接口。

### 7.8.5 USB 2.0 规范

USB 2.0 规范有 600 多页，该规范对 USB2.0 做了详尽的论述，由于篇幅太长且较为复杂，在此不作介绍。要更多了解有关此规范的信息，请参考 USB 实现者论坛网站：<http://www.usb.org>。

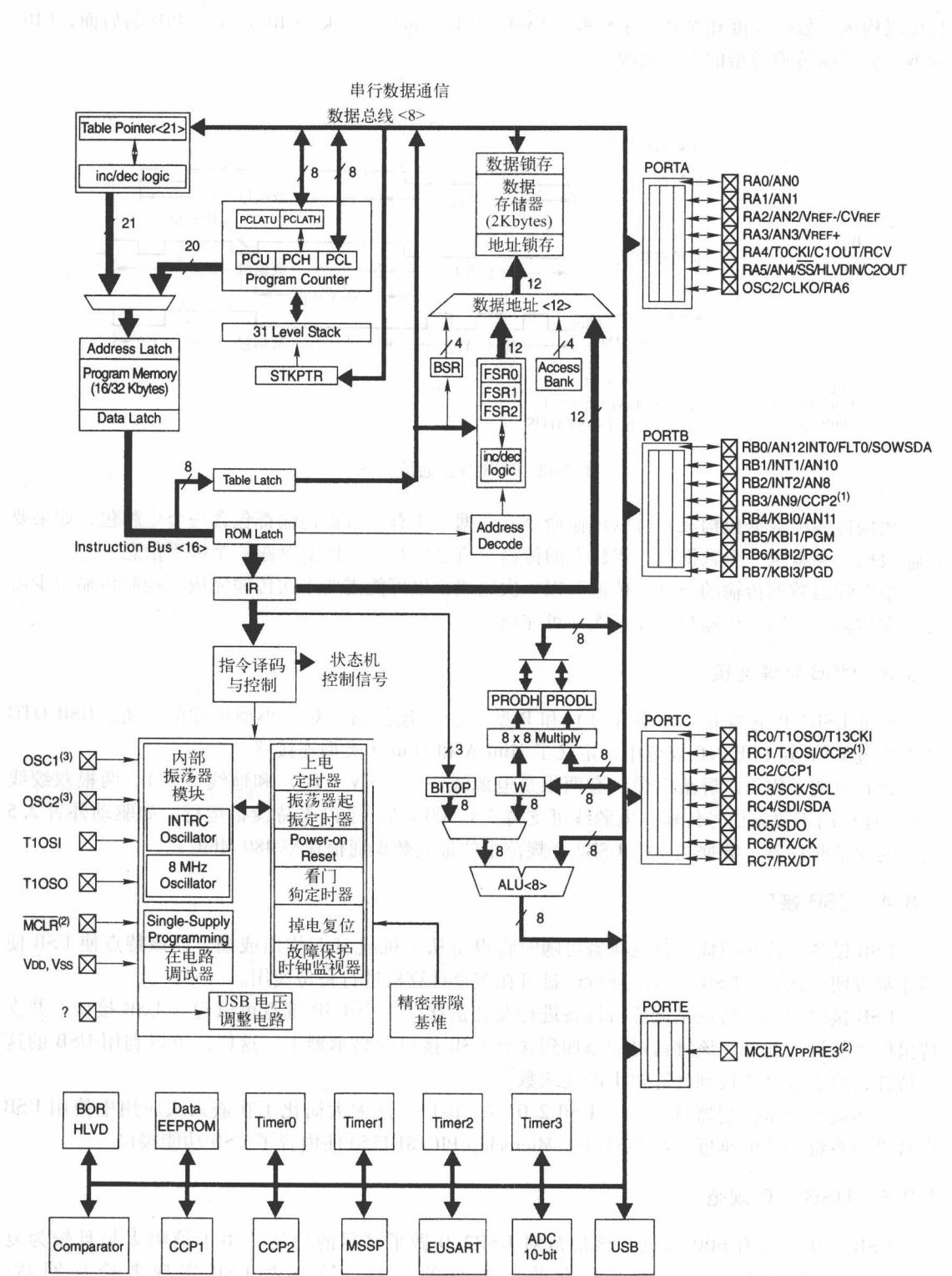


图 7-33 带有 USB 2.0 接口的 PIC18F2550 (经微芯科技股份有限公司许可使用)

## 7.9 蓝牙

从本质上来说，蓝牙是用于描述短距离（一般在 10 米以内）无线跳频协议的一个术语，该协议可以用于连接两电子设备。这些设备称作是支持蓝牙功能的设备。蓝牙原本是为了解决红外和电线同步系统间存在的问题而设计的，是通过整合一个非常小的射频模块到具有蓝牙功能的设备来解决的。

蓝牙由两个部分定义，即蓝牙规范和蓝牙应用。蓝牙规范描述了基本的蓝牙构架。蓝牙应用描述了蓝牙技术怎么使用，也就是说，怎样使用规范的不同部分实现想要的蓝牙设备功能。

当电脑、娱乐系统或者电话被使用时，它们将组成一个电子设备网络。这些设备使用不同的电线、电缆、射频和红外等相互通信，并使用差异更大的连接器、插头和不同类型的协议。蓝牙是无线的、自动的，能够简化网络的实现。

### 7.9.1 蓝牙构架

实际应用中，已经有两种方法可以不使用连线。一种是使用红外线在组件之间传递信息，红外线经常用于电视与 DVD 之间，使用 IrDA 标准进行通信。对于大部分的这种设备，红外线使用最简单的数字模式。信号是用快速的开关脉冲表示的，将数据从一个点传送到另外一个点。

另一种可以替代电线的是同步电缆线，它比红外线更麻烦。例如，通过电缆线将一台便携设备连接到一台计算机，通过一个按钮可以使两个装置的数据同步。同步电缆是一种实用的技术，当确认每台设备使用电缆的正确性以及进行对接时，将会十分繁琐。

蓝牙正是为了解决红外和电缆线同步问题而设计的，主要的硬件发起厂家主要包括 Intel、Siemens、Freescale、Ericsson 及其他厂家，这些厂家一起制定了蓝牙规范。从应用者的角度分析，蓝牙有如下 3 个主要优点：

- 是无线的；
- 不算太贵；
- 不需要过多考虑。

### 7.9.2 蓝牙频率

蓝牙使用 2.4 GHz 的频率进行通信，该频段是被国际一致认可的开放频段，可用于工业、科学、医疗等设备，简称 ISM 频段，如图 7-34 所示。很多设备都使用 ISM 频段，如婴儿监视、车库门开启控制、无绳电话等。确保和一些其他设备不会互相干扰是蓝牙设备体系结构中的一个关键部分。

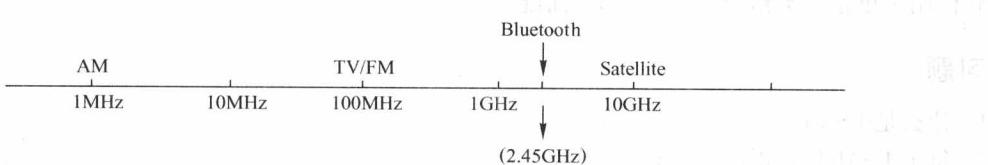


图 7-34 无线频率分配

避免干扰其他设备的方法之一是让蓝牙设备发送一个只有 1 mW 的微弱信号。相比起来，最强的蜂窝电话可以发出 3 W 的信号。低功耗限制了蓝牙设备的通信范围，大约只有 10 m 的距离，但这避免了相互干扰。房屋的墙壁不能阻挡蓝牙信号，所以使用蓝牙可以控制不同房间的多个蓝牙设备。

蓝牙使用 2.4 GHz 的频谱在两设备之间建立通信，连接速率可达 1 Mbit/s，包括一个用于声音的信道和一个 768 Kbit/s 的数据信道，整个信道的容量为 1 Mbit/s，通信数据头部和握手信息约占其中的 20%。在美国和欧洲，使用频段为 2400 ~ 2483.5 MHz，以及 791MHz 射频（Radio Frequency, RF）通道。

数据通道以每秒 1600 次的频率在 79（或 23）个频道间变频，每个频道分成 625 ms 的时隙。使用蓝牙技术建立的微微网（Piconet）包含 1 个主设备和至多 7 个从设备，主设备使用偶数时隙发送数据，从设备使用奇数时隙发送数据；一个数据包可以达到最多 5 个时隙的长度，包内数据最多可以达到 2745 位长。

蓝牙使用跳频扩频技术来抑制相互之间的干扰。在该技术中，一个设备可以使用设计范围内 79 个随机选择的独立频率，经常从一个频率中跳转到另一个频率。对于蓝牙来说，发送器每秒改变频率达 1600 次，每一信道允许的时隙为 625 ms，这就意味着可以有更多的设备能够利用这些可用的有限射频时隙。因为每一蓝牙发送器自动使用扩频发送，这样几乎不可能有两个发送器在同一时刻使用同一频率进行发送。该技术减少了使用便携电话或婴儿监视器干扰蓝牙设备的风险，因为在特定频率上的任何冲突也只能仅仅持续 1 s 中的一个小的时间片。

当蓝牙设备进入互相的有效范围以内时，会自动进行电子对话，以确定它们是否有数据需要共享，或者是否有一个设备需要控制另外一个设备。不需要用户按键或给出指令，电子对话便能够自动启动。一旦会话建立，这些蓝牙设备就建立起了一个网络。蓝牙系统创建了一个个人区域网（Personal-Area Network, PAN），或者微微网，该网络可以充满整个房间，所覆盖的距离可能还不如手机与耳机之间的距离远。微微网一旦建立，其成员会随机一致跳频，以使它们能继续保持通信，并可以避免与工作在同一房间内的微微网发生冲突。

### 7.9.3 蓝牙网络

制造商针对每一蓝牙设备进行编程，以允许具有特定地址的蓝牙设备进入已经建立的特定类型设备的地址范围内。例如，无声电话在基站内有一个蓝牙发射器，在手持设备上也有一个蓝牙发射器，当基站开启时，便发出射频信号，寻求从具有特定地址范围内的单元发出的响应信号；由于手持设备的地址正好在该范围内，它便向基站发出响应，这样就建立起一个 PAN 网络；PAN 网络有一个主设备以及至多 7 个从设备，现在，即使一个设备收到了来自其他系统发出的信号，它会忽略它，因为该信号不是来自于网络内部。

一个起居室可能建立了 3 个 PAN 网络，每一个网络中的设备都知道该网络中发送器和接收器的地址。因为每个网络每秒都会变频一千多次，任意两个网络之间便不可能在相同时间工作在相同频率。即使发生了这种情况，也仅仅会在 1 s 中的一小时间片上发生网络冲突。同时，有软件用于更正此类错误，并统计出错信息。

## 习题

1. 什么是 USART？
2. 每个 USART 字最多有多少位？
3. 为什么使用奇校验代替偶校验？
4. 为什么在 USART 的接收逻辑中使用缓冲器？
5. Freescale 公司实现的 USART 称为什么？
6. 识别 SPI 总线的信号线。
7. 请描述 CAN 网络的基本应用。
8. 使用 LIN 的基本原因是什么？

9. 请列出 I<sup>2</sup>S 的主要应用。
10. I<sup>2</sup>C 中的总线信号有哪些？
11. 在产品设计中使用 IrDA 原因是什么？
12. 在 IrDA 堆栈中有多少层？
13. 是什么使得 USB 变得如此流行？
14. USB 支持什么样的数据传输模式？
15. 对于 USB 接口来说，NRZI 意味着什么？
16. 蓝牙设计解决什么问题？
17. 跳频是为了解决什么问题？

## 参考文献

- Automotive controller area network (CAN) applications.* www.freescale.com. Freescale Semiconductor. September 2006.
- CAN specification, Version 2.0.* September 1991. Robert Bosch GmbH.
- Core Specification V2.1 + EDR.* July 2007. http://www.bluetooth.com
- Impact of new bus technologies.* National Instruments Presentation, NIWEEK. August 2000.
- Intel Automotive introduction to in-vehicle networking.* 2008. www.intel.com. Intel Corporation.
- Irazabal, J. M., and Blozis, S. *I<sup>2</sup>C bus overview, Philips Presentation, DesignCon 2003.* January 2003.
- Layout, Jalia, and Curt Franklin. June 2000. http://electronics.howstuffworks.com/bluetooth1.htm
- Network CAN.* 2008. Renesas Technology.
- PC16550D universal asynchronous receiver/transmitter with FIFOs data sheet.* National Semiconductor Corporation. June 1995.
- PIC18F2420/2520/4420/4520 data sheet 28/40/44-pin enhanced flash microcontrollers with 10-bit A/D and nanoWatt technology.* Microchip Technology. July 2007.
- Simple infrared remote control reference design, application note AN024001-1105.* 2008. ZiLOG Corporation.

# 模数转换

- 本章目标：基本认识模数转换
- 主要内容：

  1. 模数转换的基本原理
  2. 不同的转换算法及实现
  3. 如何处理输入信号
  4. 系统接口设计时的注意事项

## 8.0 模数转换

从日常的温度到一个人的体重，甚至是我们在家庭影院音响听到的声音，这些都是现实世界的模拟信号。而微控制器，类似电脑主机，却工作在数字领域。本章将介绍专门用于将现实世界的模拟信号转换为可供微控制器使用的数字信号的基本知识。

### 8.1 模数转换概述

信号可以分为两种不同的类型，即模拟信号和数字信号。模拟信号  $x(t)$  可以被定义在连续的时间域内，数字信号  $x(n)$  可以用来表示离散时间域内的一个数字序列。离散时间信号  $x(n)$  中的参数  $n$  是一个整数，表示采样周期  $T$ （通常指 1 s 内的采样次数）。那么，离散时间信号  $x^*(t)$  可以由连续时间信号  $x(t)$  来表示：

$$x^*(t) = \sum_{-\infty}^{\infty} x(n) \delta(t - nT) \quad (8-1)$$

其中：

$$\delta(t) = \begin{cases} 1, & t=0 \\ 0, & t \neq 0 \end{cases}$$

图 8-1 为一个实际的模数转换 (analog-to-digital, A/D) 器，该模数转换器将  $x(t)$  转换成离散时间信号  $x^*(t)$ ，其中的采样值在有限精度内表示。每一采样值用一个相应的数字编码近似表示（也就是说， $x(t)$  被转换成为一个有限精度的量化序列  $x(n)$ ）。

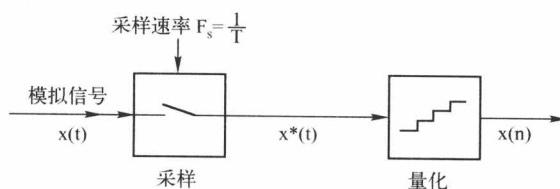


图 8-1 一般 A/D 转换器的处理过程

按照采样频率准则可将大多数 A/D 转换器分为两类，即“奈奎斯特率”(Nyquist rate) 转换器和过采样转换器。比如 SAR 或 FLASH 为奈奎斯特率转换器，而  $\Sigma - \Delta$  为过采样转换器。奈奎斯特率转换器的最高频率略低于奈奎斯特频率，即  $f_N = f/2$ ， $f$  表示采样频率。而过采样转换器拥有更高的采样频率，即  $f_N \ll f$ 。

图 8-2 说明了传统的 A/D 转换过程，即将模拟输入信号  $x(t)$  转换成数字编码序列  $x(n)$  的过

程，该转换过程的采样频率为  $f_s = 1/T$ ， $T$  表示采样周期。因为公式(8-1)中的  $\delta(t - nT)$  是以时间  $T$  为周期的周期函数，可以用傅里叶序列来表示。



图 8-2 传统的 A/D 转换过程

公式(8-1)则可以被解释为采样动作(采样函数)，等价于使用具有  $0$ 、 $f_s$ 、 $2f_s$ ……频率的载波信号对输入信号进行调制。被采样的信号可以表示为频域内原始信号组成部分和由整数倍采样频率调制的信号频率的总和。如公式(8-2)所示：

$$x^*(t) = \sum_{n=-\infty}^{\infty} x(t)\delta(t - nT) \quad (8-2)$$

因而，频率高于奈奎斯特频率(采样频率的2倍)的信号不能被正确地转换出来，并且这些信号在基带内会生成新的信号，而这些信号在原始信号中并不存在，这一现象被称为混叠(aliasing)。

为了防止混叠，输入信号需要通过一个频率高于奈奎斯特频率的低通滤波器进行预处理。这种滤波器也被称为抗混叠滤波器。它对需要的频带(基带)具有平坦的响应，对高于奈奎斯特频率的频带进行足够的衰减，使它们低于噪声基底(Noise Floor)。由于这种模拟抗混叠滤波器在带宽控制和输入信号相位失真等方面存在限制，所以需要高性能的抗混叠滤波器，以获得高分辨率和最小失真。但抗混叠滤波器对高分辨率的要求会增加一些不必要的设计成本。

除了抗混淆滤波器，还需要采样保持电路。尽管模拟信号是不断变化的，但采样要保持电路的输出在两次采样之间必须保持常量，这样使转换器有充足的时间将采样信号和给定的一组参考值进行比较，这样信号才能被正确地量化。如果采样保持电路的输出在采样周期  $T$  内发生变化，就会影响 A/D 转换器的性能。

每一个参考值对应一个数字编码。根据比较的结果，数字编码器会生成一个与输入信号最接近的编码。这种转换器的分辨率取决于预先定义的参考值的数量和间距。对于高分辨率的奈奎斯特采样，建立这样的参考电压是一个严峻的挑战。

例如，一个 16 位 ADC，就是高精度 ADC 的标准，需要  $2^{16} - 1 = 65535$  个不同的参考值。如果转换器的输入只有 2 V 的动态调节范围，那么这些值的间距只有 30 mV，这超出了超大规模集成电路(VLSI)技术中元器件的匹配容差的限度。一些新技术，如激光微调或自我校准，可以用来增加超过正常元件容差的奈奎斯特转换器的分辨率，然而，这些方法将会增加设计的复杂性，并会增加电路的面积和成本。

## 8.2 换能器

我们生活在一个模拟的世界(目前忽略了量子机械效应!)，如果现在进行天气预报，则必须把温度、风速、方向、气压等参数转换为计算机能处理的数字编码。在电路中，通过换能器获得的模拟电压值最终会被转换成由 1 和 0 组成的数字串。

常用的换能器有许多不同的类型，图 8-4 列出了一些常见的类型。

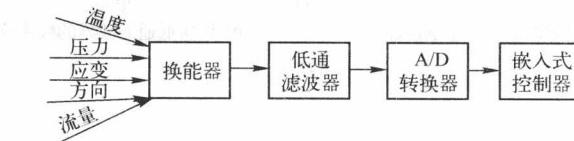


图 8-3 换能器接口

换能器可以将能量从一种形式转换到另一种形式。传感器是换能器的一种类型。对于微控制器的电子应用，它们将能量转换为相应的感应电压（或电流），霍尔效应开关便是其中一个典型的例子。

当使用传感器时，必须要注意该传感器只对被测属性十分敏感，不应该对其他属性敏感，并且传感器本身不应该影响被测量的属性。接近这些目标才是较为理想的传感器。图 8-5 展示了温度传感器具有理想的温度 - 电压线性输出关系。

机电	电声	光电	热电
传感执行器	麦克风	光敏二极管	热电偶
电流计	耳机	LED	热敏电阻
伺服机构	扩音器	太阳能电池	

图 8-4 换能器类型

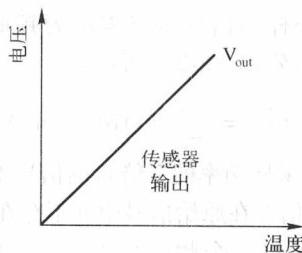


图 8-5 线性传感器输出

### 8.3 低通滤波器

低通滤波器用于限制输入模拟信号的频率，以便进行量化。典型的 A/D 转换器（除  $\Sigma-\Delta$  外）采用低通模拟滤波器，并使用两倍的奈奎斯特率进行采样。图 8-6 所示为无源低通滤波器。

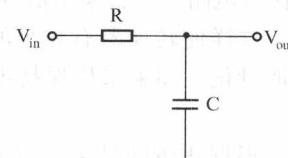
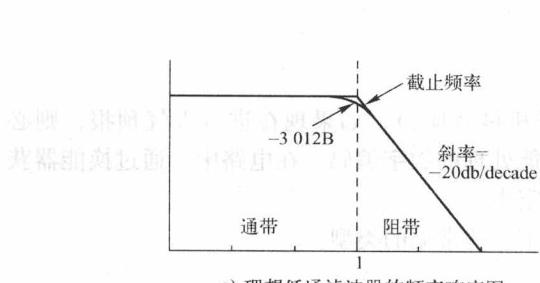
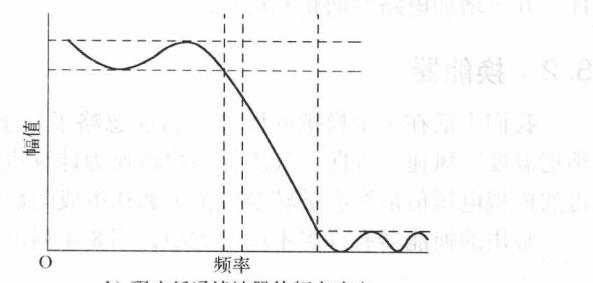


图 8-6 无源低通滤波器

图 8-7a 给出了一个理想低通滤波器的响应曲线。截止频率的临界点  $f_c$  被定义为在低于通带频率的  $-3 \text{ dB}$  处。应该注意，通带频率在  $-20 \text{ dB}$  的斜率衰减，这可以通过多极设计来改善。多极意味着多个 RC 网络，图 8-7b 显示的频率响应看起来很像是实际电路中的频率响应，需要注意通带频率响应中的波动。



a) 理想低通滤波器的频率响应图



b) 现实低通滤波器的频率响应

图 8-7

### 有源滤波器

图 8-8 给出了一种采用运算放大器实现的有源低通滤波器，电容 C 和反馈电阻  $R_2$  并联。

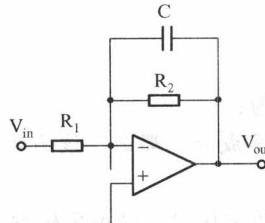


图 8-8 有源低通滤波器

该电路临界频率的计算可以由下式计算出：

$$f_c = \frac{1}{2\pi R_2 C} \quad (8-3)$$

## 8.4 采样

A/D 转换过程的关键部分是采样时间。需要转换的模拟量的采样频率对最终输出数字量结果的准确性有很大影响。图 8-9 显示了对季节性降雨量这种模拟信号的采样值。

图 8-10 中绘制了收集到的 4 个采样值，每个季度一个，用以确定每年的平均降雨量。可以用线将它们连接起来表示全年的平均降雨量，由此，可以推断出每月的总降雨量，如图 8-11 所示。

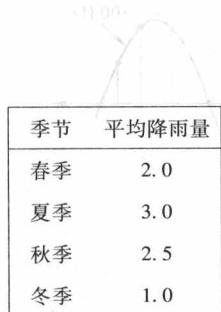


图 8-9 季节降雨量

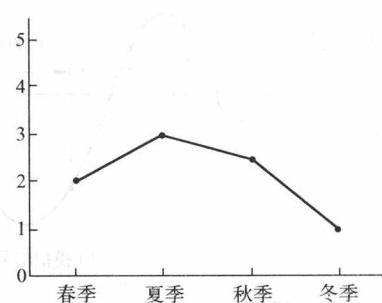


图 8-10 季节降雨量曲线

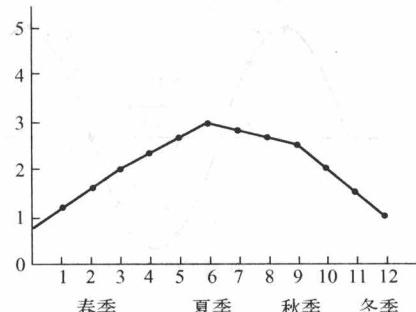


图 8-11 月降雨量曲线

从图 8-11 所给的数据来看，月平均降雨量在过去的一年里连续平滑变化。这是对每一季节降雨量抽样一次的结果。然而，它真的准确吗？从月降雨量曲线图中（见图 8-13）可以看到，图 8-12 中月降雨量的数值和图 8-11 中的数值是不一致的。事实上，图 8-11 没有正确地预测出 7 月份降雨量的显著变化。

这些问题都源于采样频率的不同。结果越要求准确，则越需要更高的采样频率，这样才能让数字输出更加精确。如图 8-13 所显示，7 月份月的降雨量有很明显的下降。

月份	降雨量	月份	降雨量
1	2.0	7	3.0
2	2.5	8	4.5
3	3.0	9	3.0
4	4.0	10	2.5
5	4.0	11	2.0
6	3.5	12	1.5

图 8-12 月降雨量表

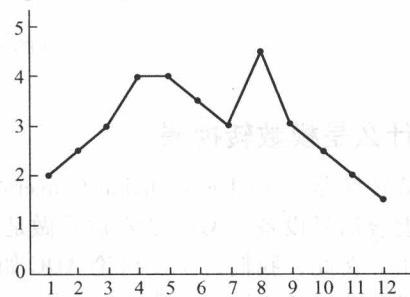


图 8-13 月降雨量曲线图

## 8.5 香农采样定理

香农对香农采样定理的原始描述为：

如果一个函数不包含高于  $\omega$  的频率成分，那么，该函数可以通过由间距为  $1/2\omega$  的一系列坐标点唯一确定。

换句话说，当对信号进行采样或数字化时，最小采样频率必须是信号频率的 2 倍。

根据香农的理论，采样频率最小不小于模拟信号的两倍是很必要的。这就是我们所熟悉的奈奎斯特频率。在这个频率下，信号可以无失真地还原。如果采样频率小于奈奎斯特频率，那么这就是所谓的欠采样信号。当一个欠采样信号被重新转换成模拟信号时，原本不属于原始信号的频率将会出现，更高频率的正弦波形现在有了一个混叠，而更低的正弦波形则不会出现在原始频率中。

图 8-14a 显示了对一个正弦波模拟信号每间隔  $T_s$  采样一次，它的采样周期就等于  $1/F_s$ 。图 8-14b 所示的正弦波频率被定义为 100 Hz，相关的时间间隔尺度为 0.005 s，根据奈奎斯特定理，采样频率必须大于 200 Hz（两倍于模拟信号的频率）。

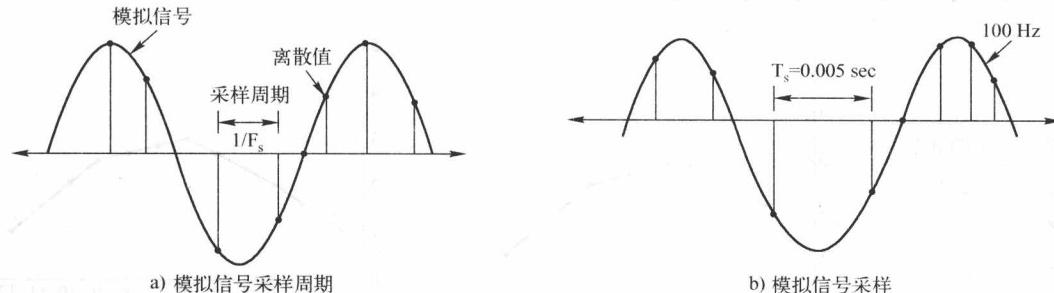


图 8-14

图 8-15 显示了一个基本的模拟信号，采样频率不满足要求，则可以从采样点中获取一个更高频率的信号。这个更高频率的信号就是混叠，因为它没有在原始信号中出现。

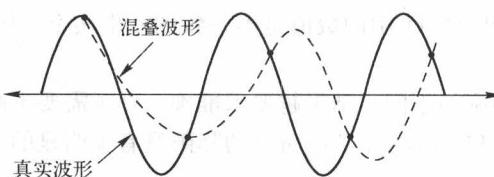


图 8-15 混叠信号

## 8.6 什么是模数转换器

模数转换器（Analog-to-Digital Converter, ADC）同时拥有模拟和数字功能，因此它被认为是一种混合信号设备。ADC 通常被看做是一个非常简单的设备，提供输入电压或输入电流的数字输出。在此，我们将主要讨论 ADC 如何将输入电压转换为二进制编码的数字量输出。

一般来说，ADC 是一个量化器，有一个模拟的参考电压用来和输入的电压进行比较，数字输出代表了输入电压相对于参考电压的分数。输入/输出传递函数可以用以下公式表示：

$$\text{Output} = 2^n \times G \times A_{IN} / V_{REF}$$

$G$  代表增益，通常为 1，所以该公式经常省略这个变量。然而，有些生产厂家，如松下半导体公司，会在 ADC 中引入不同的增益因子。

### 8.6.1 ADC 的分辨率

ADC 的二进制编码输出有  $2^n$  个编码， $n$  是转换器的位数。图 8-16 给出了一个 3 位的 ADC，其参考电压为 8 V，总共有  $8(2^3)$  个可能的编码，它们分别是：

$$2^0 = 000 \quad 2^4 = 100$$

$$2^1 = 001 \quad 2^5 = 101$$

$$2^2 = 010 \quad 2^6 = 110$$

$$2^3 = 011 \quad 2^7 = 111$$

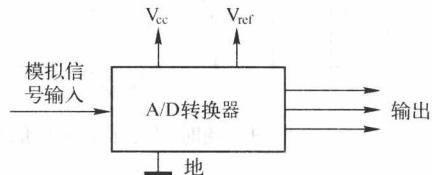


图 8-16 8 位 A/D 转换器

假设输出响应没有错误，每次在输入中增加 1 V 电压，输出代码将会加 1。这意味着在这个例子中最低有效位代表 1 V，这是该转换器所能分辨的最小增量值。

在分辨率  $n$  相同的情况下，减小参考电压才可以测量更小的电压范围，从而获得更高的测量精度。用 0.8 V 的参考电压，则最低有效位将代表 100 mV。在不使用更高精度 ADC 转换器（高精度意味着高成本）的情况下，该方法可以提高转换精度。

### 8.6.2 LSB 和 MSB 定义

最低有效位 LSB 和最高有效位 MSB 仅简单地表示为二进制代码中的最低位和最高位所代表的值。LSB 的权重为  $1(2^0)$ 。对于一个  $n$  位的编码，MSB 的权重为  $2^{(n-1)} = 2^n / 2$ 。图 8-17 从右到左显示为 LSB 到 MSB，是 8 位转换器常用的一种工程约定。

MSB								LSB
$2^{(n-1)}$	$2^{(n-2)}$	$2^{(n-3)}$	$2^{(n-4)}$	$2^{(n-5)}$	$2^{(n-6)}$	$2^{(n-7)}$	$2^{(n-8)}$	
0	1	1	0	0	1	1	0	

图 8-17 MSB-LSB

因为  $1\text{LSB} = V_{REF}/2^n$ ，要实现更高的精度，可以采取的措施有两种，一是使用更高分辨率的转换器；二是采用更小的参考电压。

### 8.6.3 量化

模拟信号不能瞬间被转化成数字信号，而需要花费一定的时间来完成模拟信号到数字信号的转换，一般由采样和保持系统来完成，如图 8-18 所示：模拟信号被采样，而后被保持直到模数转换完成。

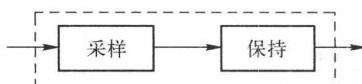


图 8-18 采样和保持

图 8-19 给出了一种典型的模拟信号波形，时间增量代表了 A/D 转换需要的保持时间。这涉及每秒的采样点数，每秒的采样点数则代表了 ADC 的转换能力。

图 8-20 中显示了信号的量化结果及每个采样周期的保持时间。从该图中可以很明显地看到，模拟信号上很多点不能被转化成数字信号，这种转换时间的限制反映了 ADC 精确表示输入信号的能力。

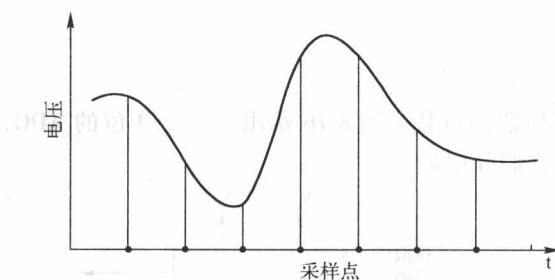


图 8-19 随时间变化的模拟信号

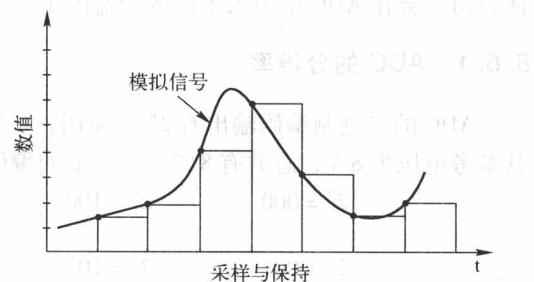


图 8-20 模拟信号的量化

ADC 的数字部分生成量化结果，这个结果反映了模拟信号的幅度。图 8-21 显示了采用 3 位 ADC 对某模拟信号进行量化的结果。3 位 ADC 可以量化 8 个值，如图 8-22 所示。

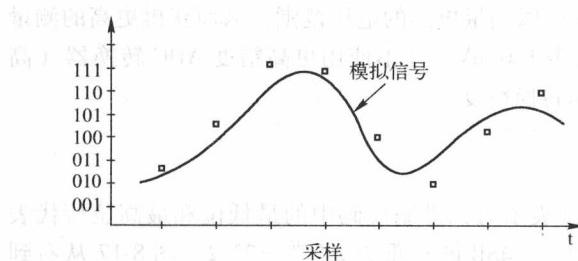


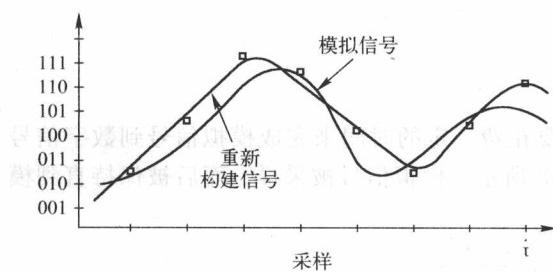
图 8-21 信号的量化

采样间隔	量化值
1	000
2	010
3	100
4	101
5	100
6	011
7	100
8	101
9	101
10	100

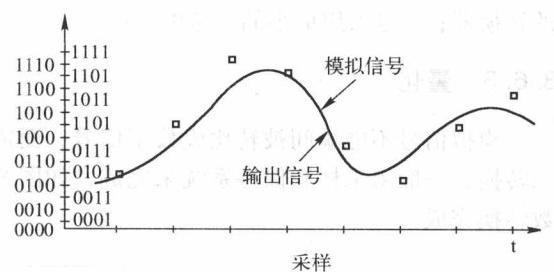
图 8-22 量化取值实例

图 8-23a 显示了使用 3 位 ADC 重新构造输入信号的结果，很明显，重新构造的信号不能很好地代表原输入信号，这是采用低分辨率进行量化处理的结果。

增加信号量化的阶数可以提高精度。在采样和保持时间减少的情况下，用一个工作在 2 倍频率下的 4 位 ADC 可以实现更多的模拟信号转换。4 位 ADC 具有 16 个量化阶数，如图 8-23b 所示：



a) 采用 3 位 ADC 的量化结果



b) 采用 4 位 ADC 的量化结果

图 8-23

#### 8.6.4 量化误差

对于简单的 3 位 ADC，当输入为 0 时，输出也为 0 (二进制 000)。当输入电压逐步向  $V_{REF}/8$

增加（即输入不再为0）时，因为输入信号的电压范围还仅由单一的输出编码来表示，输出信号仍然为0，但由于输入信号不再是0，使得误差也在加大。假如输入达到 $V_{REF}/8$ ，输出精确地与输入相对应，误差减小为0，输出从000变为001。当输入超过 $V_{REF}/8$ 时，误差再次产生。这一过程伴随着整个输入，误差波形呈锯齿状，如图8-24所示。

这里最大误差为1LSB。0~1LSB范围被称为量化不确定度（quantization uncertainty），由于任意给定编码都会对应一个模拟输入的电压范围，因此，不能确定输入电压到底是多少。最大的量化不确定度被称为量化误差（quantization error）。误差来自ADC有限的分辨率，也就是说，ADC仅能将输入分辨出 $2^n$ 个离散值，因此，转换器的分辨率也是 $2^n$ 。

因此，对于8V的参考电压（增益因子为1），一个3位ADC能把输入转化为间隔 $V_{REF}/8 = 8V/8 = 1V$ 的离散值。因此，量化误差是一种舍入误差。

0~1LSB的误差不能当作+1/2LSB误差，因此ADC引入一个偏移量来使误差范围变为+1/2LSB，如图8-25所示。输入值增加1/2LSB偏移量时，输出值将相应改变1/2LSB。当输出值从000变化到001，其输入值是在1/2LSB，而非1LSB。所有后续输出值都在1/2LSB点处变化，此时输出值的变化不再包括偏移量。

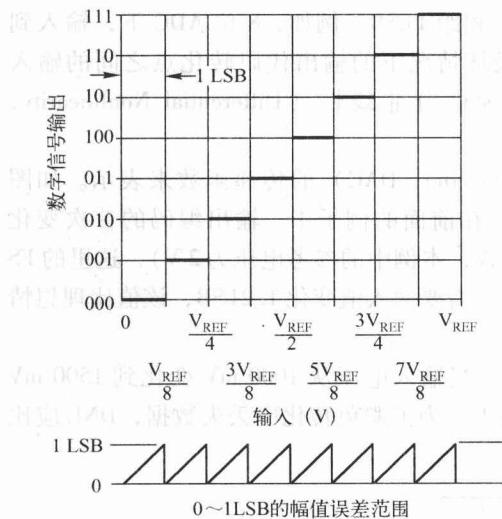


图8-24 量化误差 (NSC 2003版权许可)<sup>①</sup>

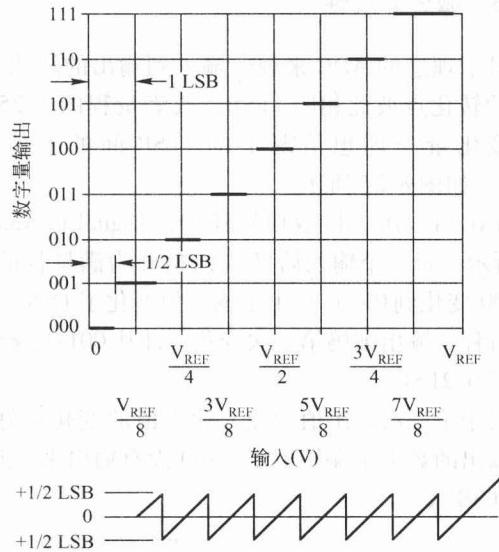


图8-25 1/2LSB偏移量误差 (NSC 2003版权许可)

当输入为0V时，输出同以前一样为000。当输入电压趋于1/2LSB时，误差随之增大，因为输入不为0，但输出仍为000。当输入达到1/2LSB时，输出由000变为001。而使输出变化的输入值不再是在1LSB，而是在1/2LSB，因此现在误差为1/2LSB。

当输入超过1/2LSB时，误差变为负值，直到输入达到1LSB时，误差为0。当输入超过1LSB时，误差随之再次增大，直到输入达到 $1\frac{1}{2}$ LSB，此时输出值增加1，误差又一次向负值方向变化，这一过程伴随整个输入范围。

要注意的是，输出值变化点与无偏移量时相比减少了1/2LSB，因此输出值第一次变化（000

① 根据National Semiconductor公司(<http://www.national.com>)的授权，本图中引用的数据进行了重新加工。

到 001) 是在  $+1/2\text{LSB}$  处, 输出值 110 到 111 的变化是发生在  $V_{\text{REF}} - 1 \frac{1}{2}\text{ LSB}$  处。当输入值超出范围时, ADC 的输出值不会像数字计数器一样循环计数, 而是始终保持满额值输出。

### 8.6.5 偏置误差

对于理想的 ADC 来说,  $q/2$  的电压输入值勉强会使输出从 0 变化到 1, 如图 8-26 所示的那样, 这种偏差被称为“零位偏置误差”(zero scale offset error) 或“偏置误差”(offset error)。当第一个转化点高于理想情况时, 此误差为正, 小于理想情况时为负。偏移误差是常量, 能被轻易地剔除, 可以用电压满量程的百分比或 LSB 表示。

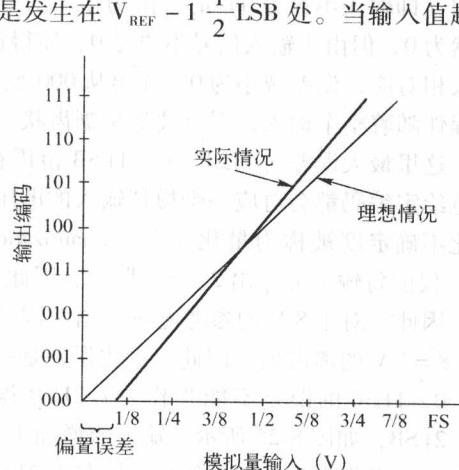


图 8-26 偏置误差 (NSC 2003 版权许可)

### 8.6.6 微分非线性

对于理想的 ADC 来说, 输入到输出的转化点正好相距 1LSB。例如, 8 位 ADC 下, 输入到输出的转化点彼此相距 1LSB, 或满量程的  $1/256$ 。最坏情况下的输出代码转化点之间的输入电压变化量与理想情况下的 1LSB 间的差值被称为微分非线性 (Differential Nonlinearity, DNL), 如图 8-27 所示。

DNL 可以用 3 位数模转换器 (Digital-to-Analog Converter, DAC) 的传递函数来表示。如图 8-27 所示, 每一个输入阶跃应该精确为满量程的  $1/8$ 。在前面的例子中, 输出编码的首次变化 (从 000 变化到 001) 是由于输入值变化了  $\text{FS}/8$  ( $250 \text{ mV}$ , 本例中的参考电压为  $2 \text{ V}$ ), 这里的 FS 是满量程。输出编码第二次变化, 即从 001 变化到 010, 需要输入值变化  $1.2\text{LSB}$ , 该值比理想情况大了  $0.2\text{LSB}$ 。

对于第三次输出值变化, 输入值的变化恰好合适。当输入电压从  $1000 \text{ mV}$  变化到  $1500 \text{ mV}$  时, 输出的数字量保持不变, 100 没有输出出来, 此码丢失。为了避免转化中丢失数据, DNL 应比  $-1.0\text{LSB}$  大。

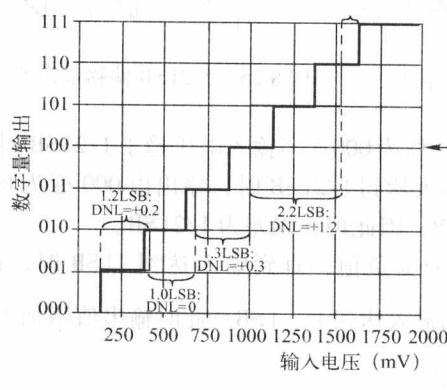


图 8-27 微分非线性 (NSC 2003 版权许可)

DNL 标志了输出电压转化一次的输入电压变化量与理想的 1LSB 间的差值, 是一静态技术指标, 与动态技术指标信噪比 (signal-to-noise, SNR) 相关。然而, 从 DNL 性能不能推测出噪声性

能，除非信噪比非常差，如同于 DNL 背离 0。

### 8.6.7 丢码

接下来要考虑的是输出中不可出现的编码。当没有输入电压能够生成一个给定的输出编码时，该编码也就不会出现在输出中，这一编码在转换中即会丢失，该编码称作丢码或遗漏码（Missing Code），如图 8-28 所示。

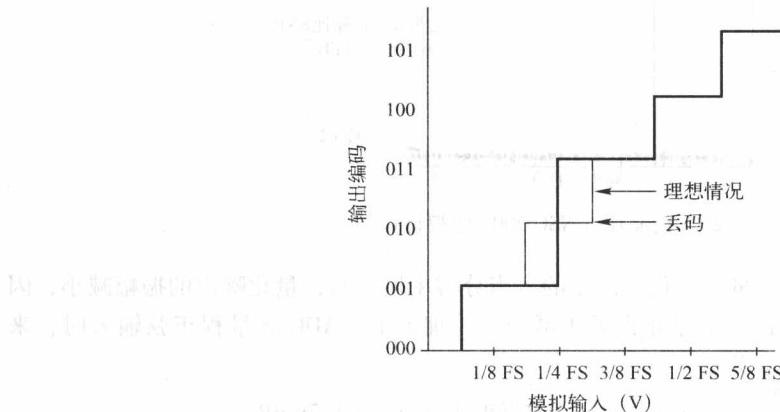


图 8-28 丢码 (NSC 2003 版权许可)

这是一个 3 位 ADC 的传递函数。第一次代码转化，即从 000 到 001，发生在输入电压为  $1/2\text{LSB}$  时，转换正确；第二次转化发生在输入电压到达  $1/4\text{FS}$  时，因此该点的微分线性误差为  $+1/2\text{LSB}$ 。第三次转化的微分线性误差为 1LSB，这些误差促使输出值从 001 直接跳到 011，使得 010 就成了丢码。

当 DNL 为 -1.0 时，则可能导致出现一个或者多个丢码。如果 DNL 小于 -1.0，意味着部分传递函数呈现负的斜率。尽管许多 ADC 数据手册中会指出“无丢码”，但在一些应用中仍要特别注意这一指标，例如伺服系统。

### 8.6.8 信噪比

信噪比（SNR）是输出信号振幅与输出噪声振幅的比值，这里的噪声不包括谐波或直流分量。 $1\text{V}_{\text{RMS}}$  信号与  $100\mu\text{V}_{\text{RMS}}$  噪声产生的信噪比是 104 或  $80\text{ dB}^{\ominus}$ 。如图 8-29 所示，噪声会以  $1/2$  的时钟频率进行整合。

通常，当频率增加时，SNR 会降低，因为 ADC 中的比较器精度会以更高的输入转换速率降低。输出时，损失的精度以噪声形式显示。

在 ADC 中，噪声主要有如下 4 个来源：

- 量化噪声。
- 转换器本身产生的噪声。
- 应用电路中的噪声。
- 抖动。

$\ominus$  RMS：RMS 表示直接度量 AC 电压中的有效值， $1\text{V}_{\text{RMS}}$  信号在一个电阻中所产生的热量与  $1\text{VDC}$  信号所产生的热量相同，通常电压的 RMS 值定义为  $\text{V}_{\text{RMS}} = \sqrt{\text{avg}(\text{V}^2)}$ 。——译者注

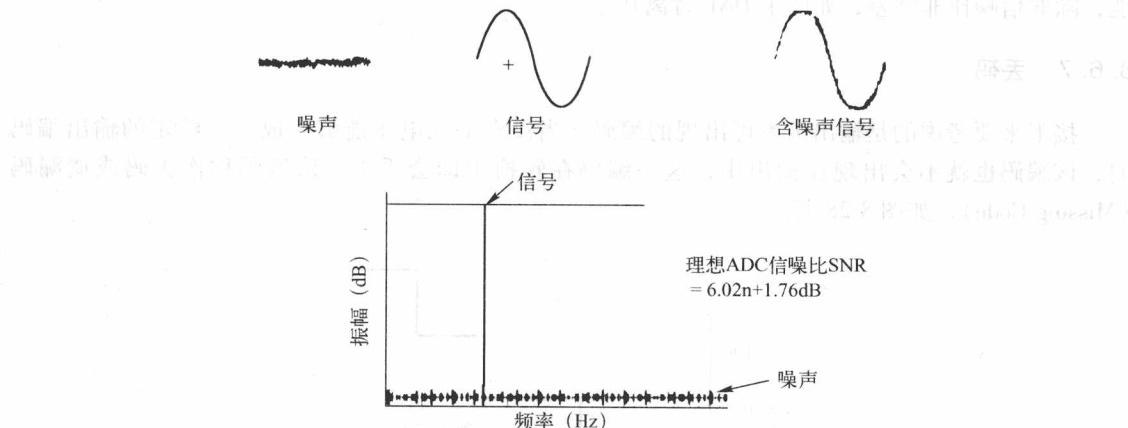


图 8-29 信噪比 (NSC 2003 版权许可)

量化噪声是在输入到输出的量化过程中产生的。当分辨率增加时，量化噪声的振幅减小，因为此时 LSB 的值变小了，同时最大的量化误差也减小了。理论上，ADC 满量程正弦输入时，来自量化噪声的最大信噪比可以定义如下：

$$20 * \log(2(N - 1) * \sqrt{6}) \text{ 或近似为 } 6.02n + 1.76 \text{ dB}$$

当输入振幅增加时，信噪比增加，直到输入接近满量程，即信噪比与输入信号同速率增加直到输入信号接近满量程。也就是说，输入信号振幅增加 1 dB，信噪比将同样增加 1 dB。这是因为，随着信号振幅的增加，输入信号增加的步长在信号振幅中所占的比例在减小。当输入振幅接近满量程时，信噪比的增加反比于输入信号振幅的增加。

## 8.7 模数转化算法

模数转化有许多不同的算法，微控制器应用中最常见的算法有以下 5 种：

- 逐次逼近寄存器型 (Successive-Approximation-Register, SAR)；
- 并行 (Flash)；
- 双积分 (Dual-slope)；
- 流水线 (Pipeline)；
- $\Sigma - \Delta$  (Sigma-Delta)。

除了  $\Sigma - \Delta$  外，其余的 ADC 算法常常要参考奈奎斯特采样器 (Nyquist sampler)，其采样频率一般是最输入频率的两倍。它们与模拟低通滤波器一起工作，来限制输入到 ADC 的信号的最大频率以及采样和维持电路的工作频率。

$\Sigma - \Delta$  ADC 采用一个带有噪声整形功能的低分辨率 A/D 转化器 (1 位量化器)，并采用高过采样率采样。其低分辨率的 A/D 转换器不需要使用模拟滤波器中的高精度元件，因此，该算法对于具有模数混合信号功能的设备来说十分具有吸引力。

图 8-30 中的表格显示了 ADC 的位数及当满量程为 1 V 时的转化结果的分辨率。从图表中可以看出，ADC 位数超过 16 位时，量化电压小于 0.4  $\mu$ V，该电压使用一般的模拟滤波器很难实现，需要采用昂贵的高容错部件，这就是  $\Sigma - \Delta$  转换器的优势所在。

图 8-31 显示了 ADC 的分辨率与转化速率的关系。图中清晰地显示出当采样速率 (每秒采样次数) 增加时，相应的分辨率就会减小，这是量化、采样及保持电路速度能力的限制结果。

图 8-32 显示了不同种类 ADC 的 7 个关键因素的比较。依据成本和采样速度的要求，不同的

ADC 拥有各自不同的优势。对于低成本高速度要求的消费类产品（如 DVD），1 位的  $\Sigma - \Delta$  转换器是最常用的选择。

ADC 的位数	$2^n$	LSB(FS=1V)
8	256	3.91 mV
10	1024	977 $\mu$ V
12	4096	244 $\mu$ V
14	16384	61 $\mu$ V
16	65536	15.3 $\mu$ V
18	262144	3.81 $\mu$ V
20	1048576	954 $\mu$ V

图 8-30 ADC 分辨率表

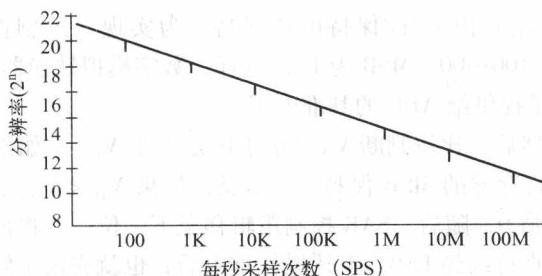


图 8-31 ADC 速率比较

	FLASH 型 (并行)	逐次逼近型	双积分型	流水线型	$\Sigma - \Delta$ 型
概述	超高速，功耗大	中高分辨率 (8 ~ 16 位)，5Msps，低功耗，小尺寸	监控直流信号、高分辨率、低功耗、良好的噪声性能	高速、几百 Msps、8 ~ 16 位、比 Flash 型功耗低	高分辨率、低速或中速、无需外部精密元件，同时 50/60Hz 抑制，数字滤波降低反混叠要求
转换方法	N 位采用 $2^N - 1$ 次比较，每增加一位比较器数量翻一番	二进制搜索算法，内部电路速度运行更高	对输入电压积分，其值与已知参考电压相比较	小型并行结构，每一级工作在 1 位或几位	过采样 ADC，抑制 5 ~ 60 Hz，可编程数据输出
编码方法	温度码编码	逐次逼近	模拟集成	数字校正逻辑	过采样调制器、数字抽样滤波器
缺点	跳码，亚稳态，高功耗，大尺寸，价格昂贵	速度局限在 0 ~ 5Msps、需要抗混叠滤波器	转换速率低、需要高精度的外部元件来满足对精度的要求	并行设计增加了耗电量和延迟	高阶 (4 阶或 4 阶以上)、多位 ADC、多位反馈 DAC
转换时间	转换时间不会随着精度的增加而改变	随着精度的增加线性增长	每增加 1 位精度转换时间翻倍	随着精度的增加线性增长	取决于数据输出速率和无噪声分辨率
分辨率	元件匹配通常将分辨率限定在 8 位	分辨率每增加 1 位，元件匹配要求将翻番	元件匹配不随分辨率的增加而增加	分辨率每增加 1 位，元件匹配要求将翻番	分辨率每增加 1 位，元件匹配要求将翻番
尺寸	$2^N - 1$ 个比较器，芯片尺寸和功耗随精度增加而指数上升	芯片尺寸随精度的增加线性增加	芯片尺寸不会随精度的增加而改变	芯片尺寸随精度的增加线性增加	芯片尺寸不会随精度的增加而改变

图 8-32 模数转换器的比较

### 8.7.1 逐次逼近

逐次逼近寄存器型 (SAR) 模数转换器 (ADC) 是常用的一种转换器结构，特别适用于中高分辨率并且采样率小于 5Msps (mega samples per second) 的情况。SAR ADC 的分辨率一般为 8 ~ 16 位，具有低功耗、小尺寸等特点。这些特点使其应用范围很广泛，如便携式电池供电仪器、笔输入量化器、工业控制以及数据/信号采集等。

顾名思义，SAR ADC 实质上是实现一种二进制搜索算法。所以，当内部电路运行频率在数兆赫兹 (MHz) 时，由于逐次逼近算法的缘故，ADC 采样速率仅是该数值的几分之一。

### 8.7.2 SAR ADC 结构

尽管 SAR ADC 的实现方式千差万别，但其基本结构非常简单，如图 8-33 所示。模拟输入电压 ( $V_{IN}$ ) 由采样/保持电路保持。为实现二进制搜索算法，N 位寄存器首先设置在中间刻度（即：100…00，MSB 为 1）。这样，数字模拟转换器（DAC）输出 ( $V_{DAC}$ ) 被设为  $V_{REF}/2$ ，其中， $V_{REF}$  是提供给 ADC 的基准电压。

然后，比较判断  $V_{IN}$  是小于还是大于  $V_{DAC}$ 。如果  $V_{IN} > V_{DAC}$ ，则比较器输出逻辑高电平或 1，N 位寄存器的 MSB 保持 1。相反，如果  $V_{IN} < V_{DAC}$ ，则比较器输出逻辑低电平，N 位寄存器的 MSB 清 0；随后，SAR 控制逻辑移至下一位，并将该位设置为高电平，进行下一次比较，这个过程一直持续到 LSB。上述操作结束后，也就完成了转换，N 位转换的结果存储在寄存器内。

图 8-34 中显示了一个 4 位转换示例。Y 轴（图中的粗线）代表 DAC 的输出电压。本例中，第一次比较表明  $V_{IN} < V_{DAC}$ ，因此，位 3 被设置为 0。接着，DAC 被设置为 0100B，进行第二次比较，因为  $V_{IN} > V_{DAC}$ ，位 2 仍保持为 1。然后，DAC 被设为 0110B，进行第三次比较，根据比较结果，位 1 被设为 0。最后，DAC 被设为 0101B，因为  $V_{IN} > V_{DAC}$ ，位 0 确定为 1。

需要注意的是，对于 4 位 ADC 需要 4 个比较周期。一般来说，N 位的 SAR ADC 需要 N 个比较周期，在前一位转换完成之前不得进入下一次转换。由此可以看出，该类 ADC 能够有效节省功耗和空间，当然，也正是由于这个原因，分辨率在 14 ~ 16 位、速率高于几 Msps 的逐次逼近 ADC 极其少见。一些基于 SAR 结构的微型 ADC 已经广泛推向市场。

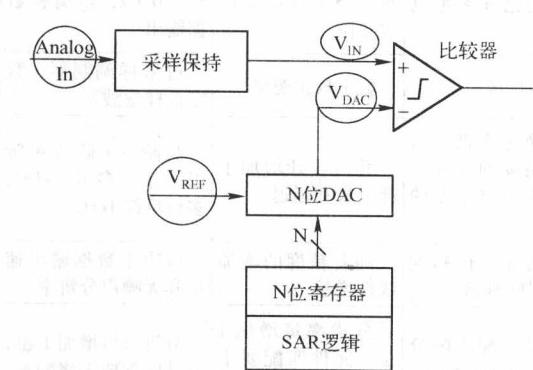


图 8-33 SAR ADC 结构图

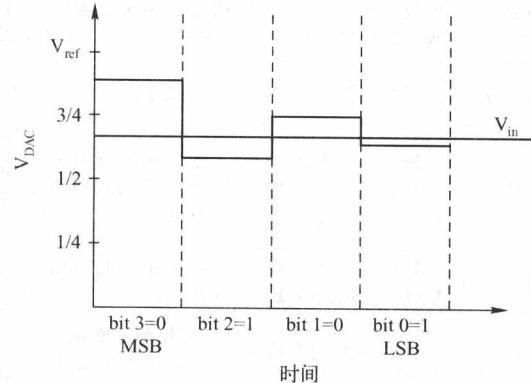


图 8-34 SAR 采样和保持

SAR ADC 的两个重要部件是比较器和 DAC。采样/保持电路可以嵌入 DAC 内，不作为一个独立的电路。SAR ADC 的速度受限于：

- DAC 的建立时间，在这段时间内必须稳定在整个转换器的分辨率以内（如：1/2LSB）；
- 比较器，必须在规定的时间内能够分辨  $V_{IN}$  与  $V_{DAC}$  的微小差异；
- 逻辑门的数目。

SAR DAC 典型应用于 16 位分辨率，相反，Flash ADC 分辨率通常限制在 8 位左右。较低的速度准许 SAR ADC 低功耗运行，例如，8 位的 SAR ADC 可在  $100 \mu\text{A}$ 、 $3.3 \text{ V}$  下以  $25 \text{ ksps}$  的转换速率运行。一款高性能的 Flash ADC 要消耗  $5.25 \text{ W}$ ，这几乎是 SAR ADC 电能消耗的 16000 倍，但其最大采样速率要比 SAR ADC 要快 40000 倍。

与其他结构相比，SAR 结构也是很经济的。假如按 1000 个的批量，一个 8 位 SAR ADC 大约花费 1 \$；相反，8 位 Flash ADC 的售价却高达 375 \$。此外，SAR ADC 封装的引脚少，更节省成本。

### 8.7.3 Flash ADC

Flash ADC 即 Flash 模数转换器，也称为并行模数转换器，在模数转换中是最快的，适用于要求带宽非常大的场合。然而，Flash 转换器消耗大量的电能，分辨率相对较低，价格十分昂贵。目前这些问题还没有很好的解决方法，也就限制了其在高频率信号处理中的应用。其主要应用包括数据采集、卫星通信、雷达处理、取样示波器以及高密度磁盘驱动器。

图 8-35 显示了一种典型的 Flash ADC 结构，内部有一个 N 位转换器，包含  $2^{N-1}$  个比较器，电路由一个电阻分压器和  $2^N$  个电阻提供参考电压。每个比较器的参考电压比紧贴其下的比较器的参考电压大 1LSB。当模拟输入电压高于所提供的  $V_{REF}$  时，比较器将输出 1，否则，比较器输出为 0。因此，如果模拟输入电压在  $V_{X_4}$  和  $V_{X_5}$  之间，比较器  $X_1 \sim X_4$  将输出 1，余下的比较器输出 0。比较器输出从 1 变为 0 的点就是输入信号小于比较器提供的  $V_{REF}$  的点。

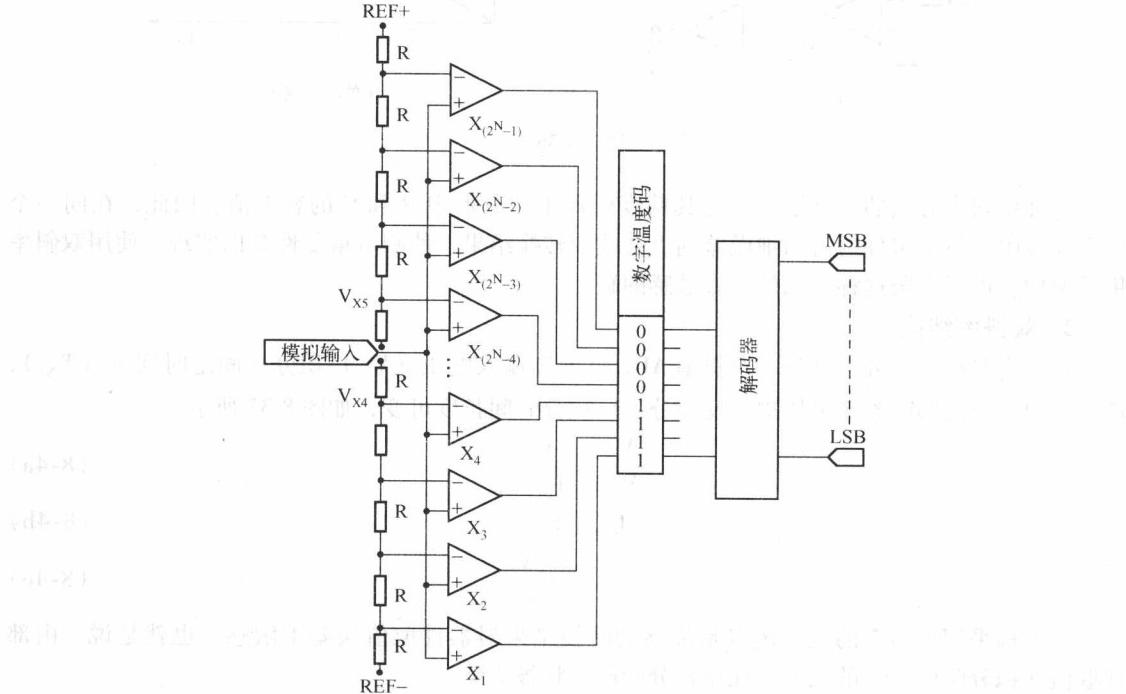


图 8-35 Flash ADC 的结构

这就是所谓的温度码，这样命名是因为它类似于水银温度计，水银柱可以一直上升到当前温度下的适当高度，随后对温度码进行解码，输出相应的数字编码。

这些比较器通常是宽带低增益级联，具有低增益特性，因为在高频下，很难同时获得高带宽和高增益。并且，这些比较器是专为低偏置（offset）电压设计的，这样每一个比较器的输入偏移量才能小于 ADC 的 1LSB。否则，比较器的偏移量可能不能满足比较器的要求，导致数字输出编码不能表示温度码。所有比较器输出之后应该由一个锁存器来保存最终结果。锁存器具有正反馈，以便最终状态可以强制为 1 或 0。

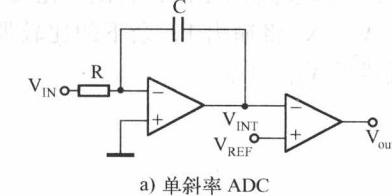
### 8.7.4 集成 ADC

集成 ADC 可以提供高分辨率，并能提供良好的线路频率和噪音抑制。这些转换器已经存在多年，但仍然适用于当今的应用。这种集成架构提供了一种新的、更为直接的、将低带宽模拟信号转换为相应的数字编码的方法。这种类型的转换器通常包含内置的 LCD 或 LED 显示驱动器，

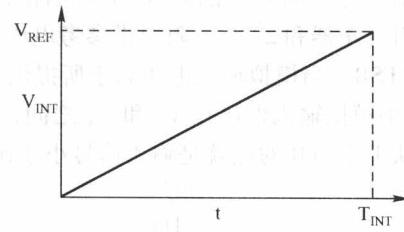
常常应用在一些便携式仪器中，如数字面板仪表和数字万用表等。

### 1. 单斜率结构

集成 ADC 的最简单的形式就是采用单斜率结构（参见图 8-36a 和图 8-36b），也称单积分结构。在这里，一个未知的输入电压被积分，并和一个已知的参考电压进行比较。这种比较器在积分过程中所花费的时间和这个未知的电压值是成正比的 ( $T_{\text{INT}}/V_{\text{IN}}$ )。因此，已知的参考电压必须是稳定的、准确的，以保证测量的精度。



a) 单斜率 ADC



b) 单斜率输出

图 8-36

这种单斜率方法的一个缺点就是其精度依赖于积分器中 R 和 C 的容差值。因此，在同一个生产环境中，每个组件的值的细微差别也会改变转换结果，使测量重复性难以实现。使用双斜率集成架构，可以克服这种对元件值太敏感的缺点。

### 2. 双斜率结构

双斜率又称双积分。首先，双斜率 ADC 将未知输入电压 ( $V_{\text{IN}}$ ) 积分一固定时间量 ( $T_{\text{INT}}$ )，接着使用一个已知的参考电压进行反积分，反积分时间长度可变，如图 8-37 所示。

$$\frac{V_{\text{IN}}}{V_{\text{REF}}} = \frac{V_{\text{REF}}}{T_{\text{DE-INT}}} \quad (8-4a)$$

$$T_{\text{INT}} = \text{Fixed} \quad (8-4b)$$

$$T_{\text{DE-INT}} \propto \frac{V_{\text{IN}}}{V_{\text{REF}}} \quad (8-4c)$$

该架构采用单斜率的主要优点是最终的转换结果对器件取值误差不敏感。也就是说，由部件取值在积分阶段引入的误差，在反积分阶段都将被去除。

相关公式为：

$$\frac{V_{\text{IN}}}{V_{\text{REF}}} = \frac{T_{\text{CHARGE}}}{T_{\text{DISCHARGE}}} \quad (8-5)$$

从这个方程中可以看出，反积分时间  $T_{\text{DISCHARGE}}$  正比于  $V_{\text{IN}}/V_{\text{REF}}$ ，如图 8-38 所示，如图 8-39 所示为一双斜率 ADC 的结构框图。

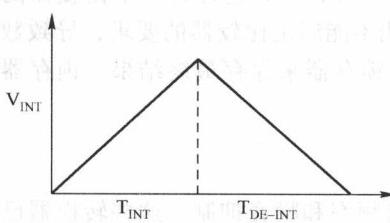


图 8-37 双斜率时序图

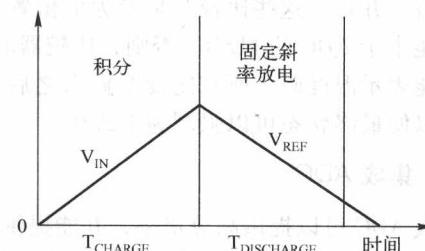


图 8-38 双斜率时序图

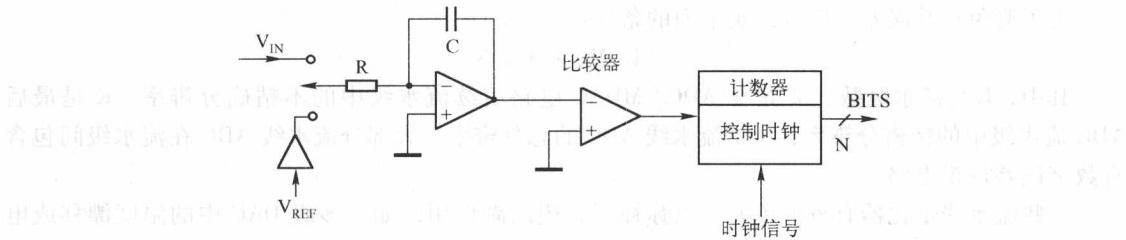


图 8-39 双斜率 ADC 框图

举例来说, 为获得 10 位分辨率, 需要积分  $1024(2^{10})$  个时钟周期, 接着反积分最多 1024 个时钟周期 (假定最多转换  $2 \times 2^{10}$  个周期)。为获得更高的分辨率, 可以增加时钟周期的数量。分辨率与转换时间之间的折中是该类型转换器内在固有的追求。对于一个给定分辨率的转换器, 适度调整电路可以加速转换时间, 但是还要注意精度和转换速率之间的平衡。

### 8.7.5 流水线 ADC

流水线 ADC 在尺寸、速度、分辨率、电源消耗和模拟设计难度上拥有最佳的平衡, 设计者很乐意选择这种结构。流水线 ADC 包括许多连续的流水级, 每一级都包含一个采样/保持器、一个低分辨率 ADC (DAC) 和一个由含有能提供增益的级间放大器组成的电路, 故又称作分级量化器。

流水线 ADC 主要应用于通信系统、图像系统和数据采集系统。在通信系统中, 其总谐波失真 (Total Harmonic Distortion, THD)、无杂散动态范围 (Spurious-Free Dynamic Range, SFDR) 和其他频域特征显著; 在基于电荷耦合器件 (Charge-Coupled Device, CCD) 的图像系统中, 其对噪声、带宽有很好的时域特性, 其快速瞬态响应保证了信号建立时间短; 在数据采集系统中, 其时域和频域特性良好 (如低杂散 [low spurs] 和高输入带宽)。

两个或两个以上的流水级可完成快速而准确的 N 位转换。如图 8-40 所示, 首先, 执行一个不精确的 M 位 A/D 转换; 然后, 使用一个至少具有 N 位精度的 DAC 进行 D/A 转换, 再转换出  $2^M$  个模拟电平, 并与输入进行比较; 最后, 差值传送到一个精密的 K 位 Flash ADC, 两个或多个输出级相结合便完成了转化。

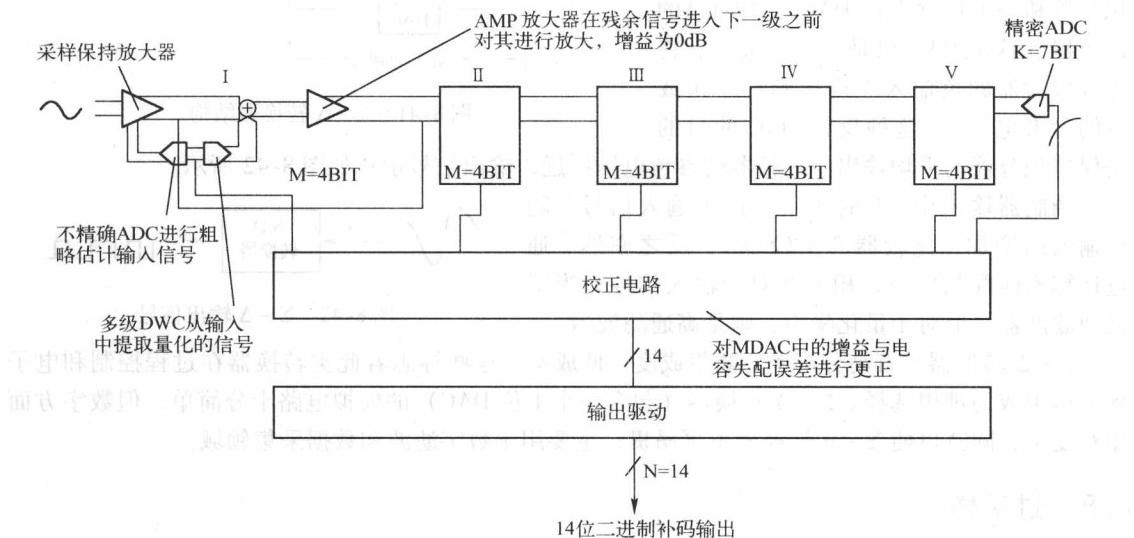


图 8-40 流水线 ADC 的功能框图

为了避免重叠误差，应该满足下面的条件：

$$(L^*M) + K > N$$

其中， $L$  是流水级数， $M$  是在 ADC/MDAC 电路后续流水级中的不精确分辨率， $K$  是最后 ADC 流水级中的精密分辨率， $N$  是流水线 ADC 的总分辨率。大部分流水线 ADC 在流水级间包含有数字误差校正电路。

一些流水线量化器有校准单元，以弥补不希望的副作用，如在多级 DAC 中的温度漂移或电容不匹配。这个数字校准通常作用于几个连续的流水级而非全部，流水级用两个相近的值在 MDAC 的输出上产生一个与  $V_{REF}$  等同的值来实现校准。所有与理想情况的偏差都需要测量弥补，这些误差都需要随后的流水级收集并保存到外存中。随后，RAM 中的误差会在正常操作中被提取出来得以弥补。其中，电容不匹配会在 MDAC 流水级中得到补偿。

新型流水线结构简化了 ADC 的设计，并且具有如下优势：

- 每个流水级有额外数位来优化重叠误差的纠正结果。
- 每个流水级都有独立的采样保持放大器，它们能释放要进入下次采样的 T/H，使得在不同流水级的采样转化能同时进行。
- 低功耗。
- 更高速率的 ADC ( $F_{conv} < 100 \text{ ns}$ , 典型)，能节省成本，可以减少开发时间。

但是流水线 ADC 也存在如下一些不足：

- 参考电路、偏置电路复杂。
- 流水线延迟，输入信号必须通过数个流水级。
- 关键锁存时序，需要将所有输出同步。
- 与其他结构相比，其对电路板布局有较高要求。
- 对误差处理的灵敏度很高，使得增益、偏移量和其他参数出现了非线性。

### 8.7.6 $\Sigma - \Delta$ 转换器

图 8-41 给出了一阶  $\Sigma - \Delta$  转换器的结构框图，包括一个差分放大器、一个积分器和一个包含 1 位 DAC 反馈回路的比较器（这个 DAC 只是一个开关，它将差分放大器的负输入端连接到一个正或负的参考电压）。这种反馈 DAC 的目的是保持积分器的平均输出与比较器的参考电压接近，输出信号示意如图 8-42 所示。

调制器输出中，1 的比重正比于输入信号。随着输入的增加，比较器产生较多 1，反之亦然。通过计算不匹配的电压，积分器对于输入信号充当了低通滤波器；但对于量化噪声，则是高通滤波器。

$\Sigma - \Delta$  转换器具有高分辨率、高集成度、低成本，这些标志着此类转换器在过程控制和电子秤应用中成为理想选择。 $\Sigma - \Delta$  转换器（包含一个 1 位 DAC）的模拟电路十分简单，但数字方面相对复杂，同时也使  $\Sigma - \Delta$  转换器不再昂贵；主要用于数字滤波和数据采集领域。

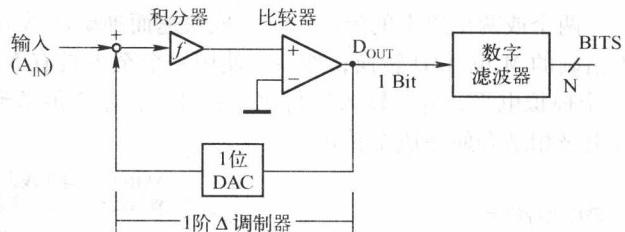


图 8-41  $\Sigma - \Delta$  转换器结构

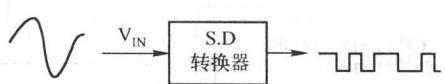


图 8-42  $\Sigma - \Delta$  输出信号

### 8.8 过采样

首先，考虑到传统的以正弦波为输入信号的多位 ADC 在频域内的传递函数，输入信号以频

率  $F_s$  进行采样。根据奈奎斯特理论,  $F_s$  必须至少是输入信号带宽的 2 倍。

当用 FFT 分析仪对数字输出结果进行观察时, 从 DC 到  $F_s/2$ , 可以明显的观察到一个单频信号和随机噪声, 如图 8-43 所示。该噪声被称为量化噪声, 源于 ADC 的输入是一个有无限可能状态的连续信号, 但数字输出是由转换精度已经决定的有限状态数目的离散函数。因此, 模拟到数字的转换过程中将会丢失一些信息, 引入了一些信号失真。这个误差幅度是随机的, 一般在  $\pm$  LSB 之间。

用信号的幅值除以所有噪声信号幅值的有效值 (Root Mean Square, RMS), 便可得到信噪比 SNR。对于 N 位 ADC,  $SNR = 6.02N + 1.76$  dB。为改善传统 ADC 的 SNR 及再生信号的准确性, 必须增加 ADC 的位数。

重新思考前面的例子, 将采样频率乘以过采样率 K, 增加到  $KF_s$ , 如图 8-44 所示。FFT 分析显示, 噪声基底下降了, SNR 和以前一样, 但是噪声能量已经分散在较宽的频率范围。Σ - Δ 转换器通过在 1 位 ADC 以后紧跟一个数字滤波器 (见图 8-45) 来充分利用这一效应, 由于大部分噪声被数字滤波器过滤, 使得噪声的有效值变小。这就使得 Σ - Δ 转换器能够实现从低分辨率 ADC (1 位) 来获得较宽的动态范围。

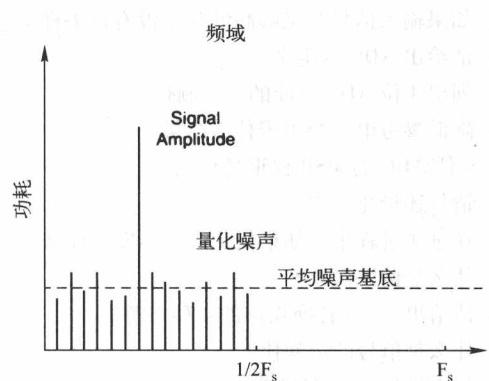


图 8-43 频域图

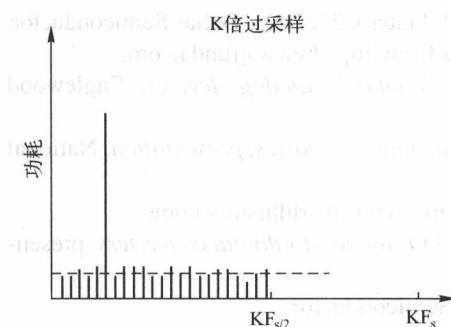
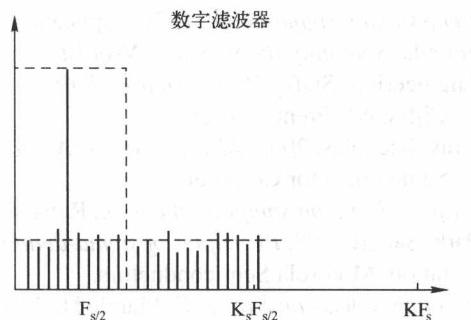
图 8-44 K 倍  $F_s$  过采样

图 8-45 数字滤波器

## 习题

1. 信号有哪两种类型?
2. 如何表示离散时间信号?
3. 主要的 ADC 有哪两种?
4. 描述什么是混叠。
5. 什么类型的滤波器能用于防止混叠?
6. 描述采样保持电路的主要功能。
7. 简述换能器的用途。
8. 使用传感器时主要考虑什么因素?
9. 一个理想的低通滤波器应该有怎样的斜坡?
10. 什么是有源旁路滤波器的临界频率?
11. 采样频率对 A/D 转换精度有何影响?

12. 香农定理中的采样频率是如何描述的?
13. 如果输入信号在足够高的频率没有被采样会怎样?
14. 请给出 ADC 的定义。
15. 列出 4 位 ADC 可能的二进制代码。
16. 降低参考电压会出现什么结果?
17. 4 位 ADC 的 MSB 权重是什么?
18. 请叙述量化过程。
19. 在量化过程中, 如果量化太低会发生什么?
20. 什么是偏置误差?
21. 请给出 A/D 转换噪声的 4 种来源。
22. 什么是信号的信噪比?
23. 列举最流行的 ADC 类型。
24. 为什么要使用不同类型的 ADC?
25. 什么情况下不适用 Flash ADC?
26. 双积分型 ADC 的主要优势是什么?
27. 比较流水线 ADC 的优点和缺点。
28. 如何确定  $\Sigma - \Delta$  转换器的转换结果?

## 参考文献

- Carr, Joseph J. 1978. *Digital interfacing with an analog world*. Blue Ridge Summit, PA: TAB Books.
- Demystifying sigma-delta ADCs*. Application Note 1870. January 31, 2003, Dallas Semiconductor.
- eFunda. *Sampling theorem and Nyquist rate*. Retrieved from <http://www.efunda.com>.
- Engineering Staff. 1986. *Analog-digital conversion handbook, analog devices*. Englewood Cliffs, NJ: Prentice Hall.
- Gray, Nicholas. 2003. *ABCs of ADCs: Analog-to-digital converter basics, presentation*. National Semiconductor Corporation.
- Nyquist-Shannon sampling theorem*. Retrieved from <http://www.worldhistory.com>.
- Park, Sangil. 1998. *Principles of sigma-delta modulation for analog-to-digital converters, presentation*. Motorola Semiconductors.
- Pipeline ADCs come of AGE*. March 21, 2000, Dallas Semiconductor.
- U.C. Berkeley. *The Nyquist-Shannon sampling theorem*. <http://ptolemy.eecs.berkeley.edu>.
- Understanding flash ADCs*. Application Note 810. October 2, 2001, Dallas Semiconductor.
- Understanding integrating ADCs*. Application Note 1041. May 2, 2002, Dallas Semiconductor.
- Understanding SAR ADCs*. Application Note 1080. March 1, 2001, Dallas Semiconductor.
- Wikipedia. *Analog-to-digital converter*. Retrieved from <http://en.wikipedia.org>.

# 数字信号处理

- 本章目标：介绍数字信号处理在嵌入式微控制器系统中的概念和应用
- 主要内容：

  - DSP 的定义
  - 基本的模拟信号技术
  - 各种模数转换类型
  - 模拟信号到数字表示的转变
  - 系统设计约束

## 9.0 数字信号处理

信号处理 (Signal Processing, SP) 通常作为一个论题被当作应用数学的一个分支。如果一个问题可以用算法进行化简，就可以在上面应用信号处理方程。信号处理包含很多重叠的应用领域，如图 9-1 所示。

信号处理有很多应用，包括机械问题、声学工程（声音识别）、生物医学工程（电脑断层扫描）、军队（雷达/声纳装置）、电话学（手机）及工业（石油开采）等。

图 9-2 展示了一个常见的消费电子项目，该项目在 MP3 播放器中使用了数字信号处理 (Digital Signal Processing, DSP)。

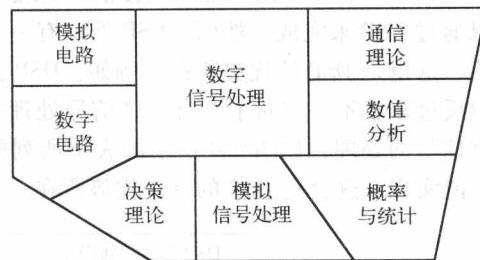


图 9-1 DSP 应用领域

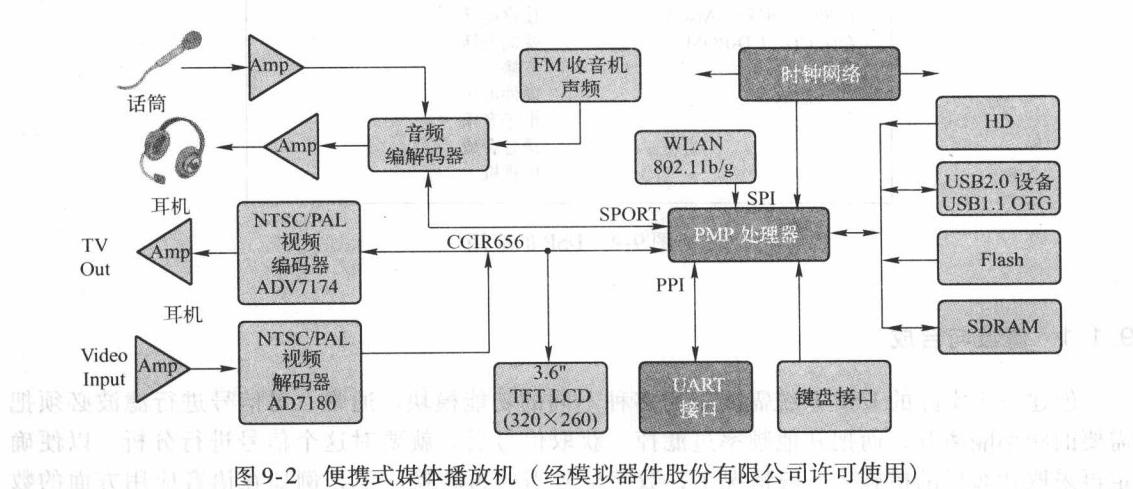


图 9-2 便携式媒体播放机 (经模拟器件股份有限公司许可使用)

## 9.1 什么是 DSP

当模拟信号（比如持续电压）转换为数字表示时，就会涉及 DSP 问题。该转换过程是通过 DAC 实现的，这已在第 8 章介绍过。信号处理方程定义了约束和限制，可获知从模拟信号中能

获得什么信息和多少信息。

图 9-3 展示了一个典型的 DSP 系统。输入信号  $X[t]$  首先通过一个抗混叠滤波器，该滤波器的设计带宽是采样频率（奈奎斯特率）的  $1/2$ 。

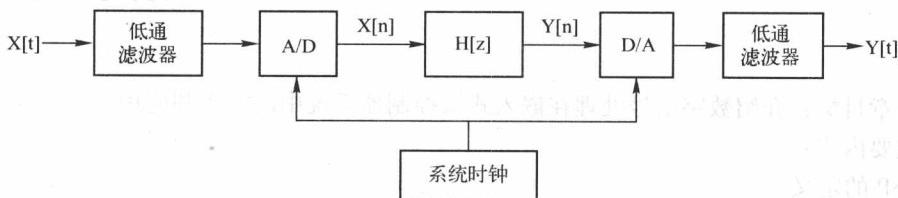


图 9-3 信号处理流程

经过滤波器滤波之后，信号由 ADC 进行数字化，其采样频率由系统时钟频率确定，生成一个离散的时间信号  $X[n]$ 。然后，信号再经过系统传递函数为  $H[z]$  的模块，产生输出信号  $Y[n]$ ，通常这一转化是在时域内进行的。接着，输出信号  $Y[n]$  会通过一个 DAC 和一个低通滤波器，生成一个连续的时间信号  $Y[t]$ 。

几乎所有可以用传统的模拟信号处理（Analog Signal Processing, ASP）方法进行的处理都可以通过 DSP 来完成。然而，DSP 处理有一个显著特点，那就是它比单纯的模拟方式应用更为广泛，这也是 DSP 的优势所在。例如：DSP 方程式可以由物理电路组成，如电感、电阻、电容等。但反过来就不一定对了，有一些信号处理方程式并不能由物理电路来表示。但这并不能制约数字信号的处理，如图 9-4 所示的表中所列的这些应用便可以由 DSP 来创建，而利用普通方法却不能实现，这就是 DSP 的真正优势所在。

DSP 创造的新产品	DSP 提高的已存在的产品
高速调制解调器	汽车发动机控制
语音和图像识别	汽车主动减震
医疗成像设备	应答机
有源噪声消除 (ANC)	便携电话
音乐 CD, CD-ROM	蜂窝电话
DVD	广播
MP3 播放器	硬件电子 电子音乐 语音合成 电视机

图 9-4 DSP 的应用

### 9.1.1 滤波与合成

创建一个实际的 DSP 系统需要采用多种不同的功能模块。通常，对信号进行滤波必须把需要的频率隔离开，而把其他频率过滤掉。获取信号后，就要对这个信号进行分析，以便确定再采取什么样的措施。一般情况下，数字信号需要再转换一下，例如在语音应用方面的数字应答机。

信号合成可以使用数字的方法生成声音信号，音乐合成器就是其中一个众所周知且十分常用的应用。相关性是信号处理的另一个方面，通过对部分信号来查找信号样本。最后，在许多应用中，还要考虑设备接口等控制方面，如硬盘驱动器。

### 9.1.2 DSP 性能

对于信号的数字化处理，有实际情况的限制，最重要的一点可能就属性能了。处理一个进程需要尽可能快地对一个信号进行转换，以满足应用的要求。很显然，一部手机不可能有大型电脑处理信号的速度。这个也与市场价格有关。手机设计必须更侧重于人们的负担能力，不论什么样的潜在信号处理特性，都可以通过工程予以实现。

### 9.1.3 模拟信号转换

我们生活在一个模拟的世界，温度的变化、水流量和风向都是模拟的过程，它们可以通过向换能器提供连续模拟信号进行测量，通常提供的是电压信号。通过信号电压与参考电压的比较，该信号可以用于启动马达或其他设备。

例如图 9-5 中展示的一个麦克风，基准电压决定换能器输出的最大电压和最小电压。它们在麦克风上标为  $V_{ref-}$  和  $V_{ref+}$ ， $V_{ref-}$  通常接地。

不同频带可以通过具有数字显示能力的 LCD 面板以数字形式显示出来。由图 9-2 所给实例可以看到，借助于 DSP 技术，音频编码器也能用于将电压信号转换为数字信号。图 9-6 所示为模拟电压与频率的关系。DSP 允许这一随时间变化的信号转换到频域，转换结果可以通过频谱分析仪显示出来。图 9-7 所示为音频 MP3 频谱分析仪。



图 9-5 麦克风

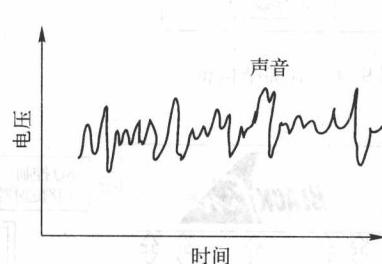


图 9-6 模拟电压与频率的关系

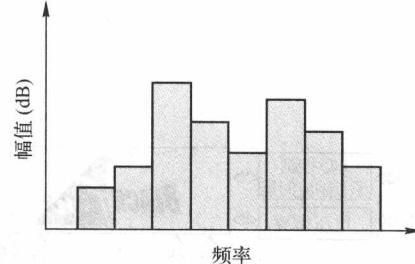


图 9-7 音频 MP3 频谱分析仪

## 9.2 DSP 控制器构架

图 9-8 展示了一个传统的哈佛构架的精简指令集原理框图。图 9-9 展示了一个更为详细的哈佛 RISC 进程，需要注意的是，其中包含了一个单独的算术功能单元，通过扩展便能够处理 DSP 算法的基本功能，提升处理速度；系统附加了一个乘法单元，用于提供 DSP 所需求的乘法累加器（Multiply-Accumulate，MAC）的支持。

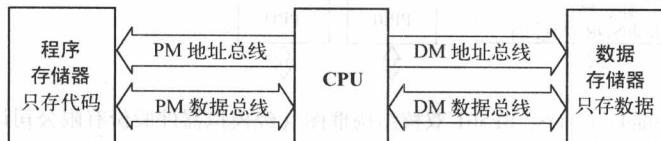


图 9-8 哈佛构架

图 9-10 是由 Analog Device 公司提供的使用 Blackfin ADSP-BF561 双核结构框图，它包含图 9-11 所示的并行 MAC 单元，该单元能够加快由乘加操作构成的离散傅里叶变换，这在本章后

面的内容中将会详述。由于使用了 MAC 单元，嵌入式单片机控制器能够执行复杂的 DSP 功能，并且可以将成本控制在很低。

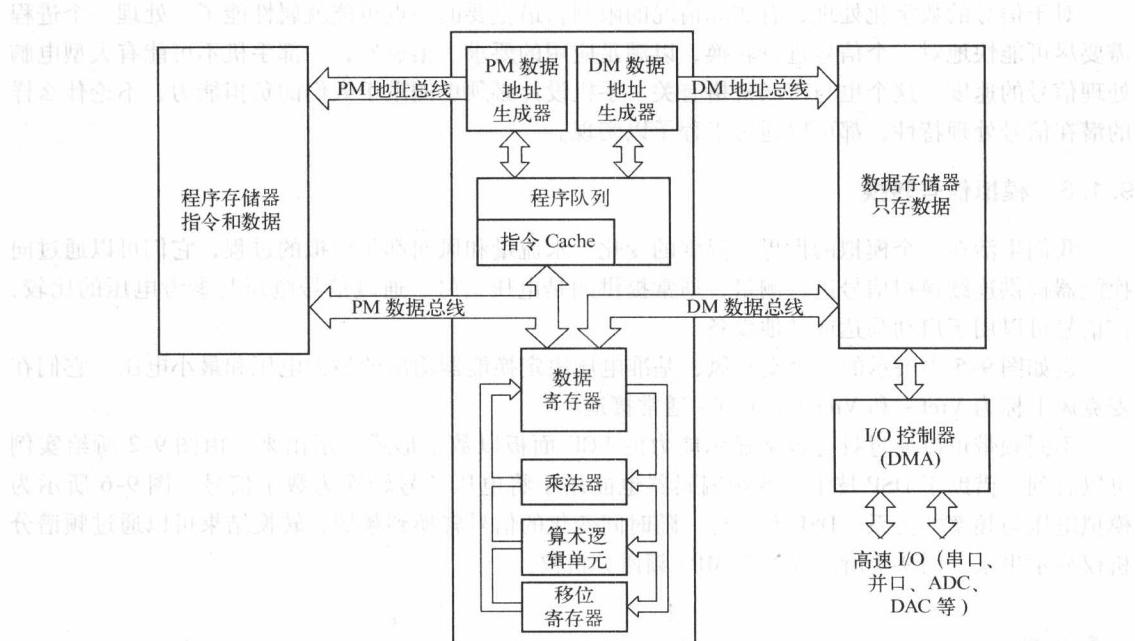


图 9-9 哈佛结构框图

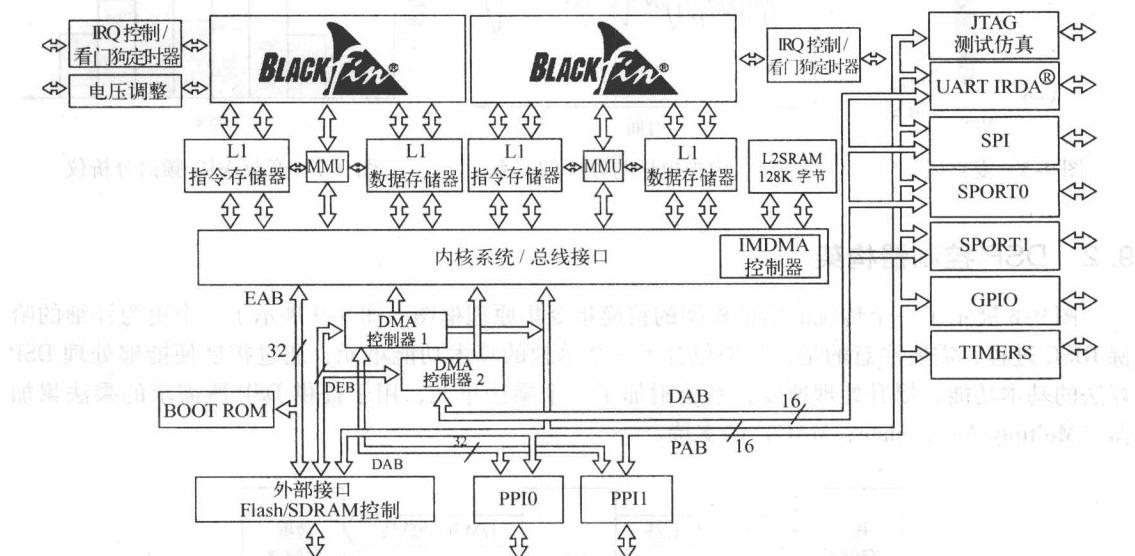


图 9-10 Blackfin ADSP-BF561 双核结构框图（经模拟器件股份有限公司许可使用）

MIPS 公司通过 MDU 功能模块将其 24 KE 系列进行了扩展，使它包含了 MAC 功能，如图 9-12 所示。这个 MIPS 24 KE 内核是 24 KE 系列 IP 核的扩展，它使一个普通的 SoC 系统芯片能够拥有功能强大的 DSP 能力。在此之前，如果要实现这样的功能，需要一个成本更高、空间更大的独立芯片来完成。

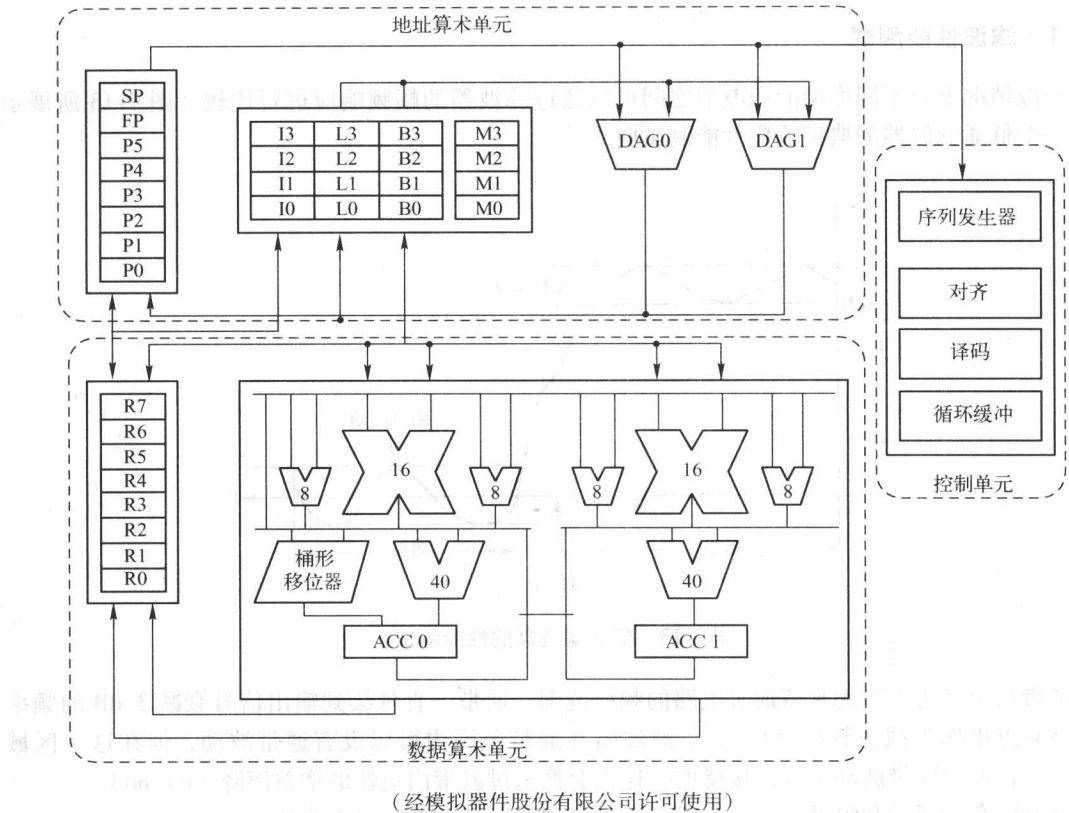


图 9-11 Blackfin ADSP-BF533 双 MAC 单元

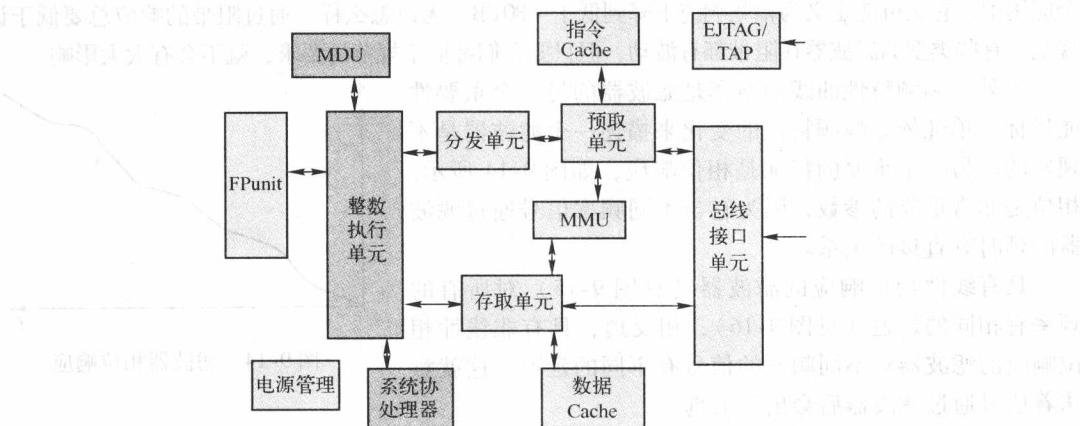


图 9-12 经 ASE 扩展的 MIPS 24 KE 内核 (经模拟器件股份有限公司许可使用)

### 9.3 模拟滤波器

在基本的数字信号处理方法广泛应用之前，主要使用的是模拟低通滤波器，包括经典的基于电阻 - 电容的无源滤波器和基于调谐放大器电路的有源滤波器。切比雪夫 (Chebyshev) 和巴特沃斯 (Butterworth) 滤波器可以通过频率特性比较，利用数学算法对滤波器进行数字化，不用无源的或基于调谐放大器的滤波电路，也可以对信号进行处理。

### 9.3.1 滤波性能测试

一般情况下，不同类型的滤波器之间可以通过滤波器的幅频响应进行比较。图 9-13 所展示的是一个低通滤波器的典型响应性能测试曲线。

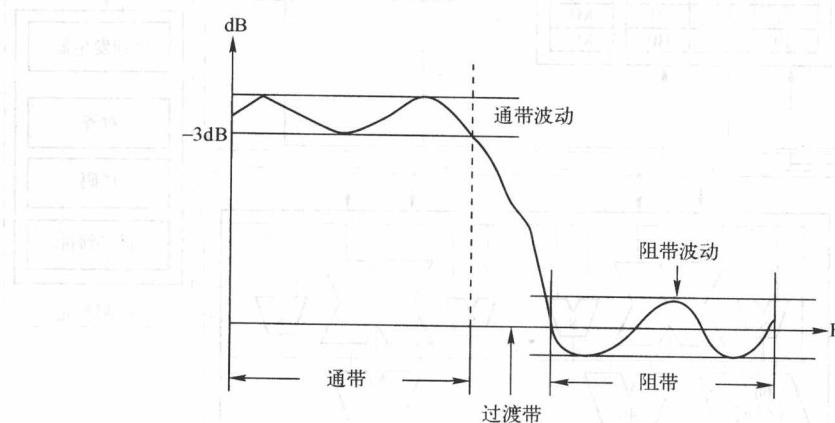


图 9-13 低通滤波器的性能测试

通带被定义为信号无衰减通过电路的频率范围。通带一直延续到输出信号衰减 3 dB 的频率点，该频点也称为截止频率 ( $f_{-3\text{dB}}$ )。滤波器在通带范围内可以没有通带波动，但在这一区域内，一定程度的通带波动是可以接受的，这也会换来过渡带内更快的增益滚降 (roll-off)。

过渡区介于通带和阻带之间，前面已经讲过这个区域的频率是下降的。

设计者会根据需要设计不同的阻带。例如：它可能定义为幅频响应下降到低于 -4 dB，而在另一个应用中，它又可能定义为幅频响应下降到低于 -80 dB。无论怎么样，通过阻带的响应总要低于设计规范。有些类型的滤波器在阻带都有波动，但只要它们都低于规范的要求，就不会有太大影响。

另外，幅频特性曲线的斜率是滤波器的另一个重要性能指标。单纯依靠幅频特性的变化来描述一个滤波器是不现实的。另一个重要的特征是相位响应，如图 9-14 所示。相位是非常重要的参数，因为它与不同频率信号通过滤波器的延时有直接的关系。

具有线性相位响应的滤波器（见图 9-15）对所有的频率有相同的延迟（见图 9-16），相反地，具有非线性相位响应的滤波器对不同频段的信号有不同的延迟，这就意味着信号通过滤波器后会出现失真。

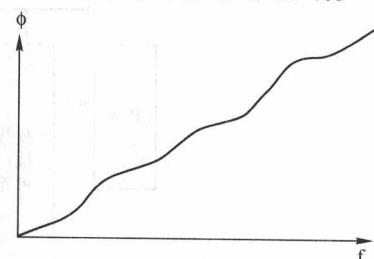


图 9-14 滤波器相位响应

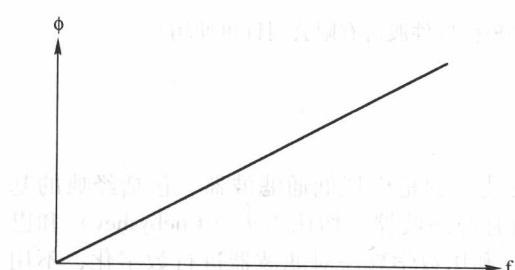


图 9-15 线性相位响应

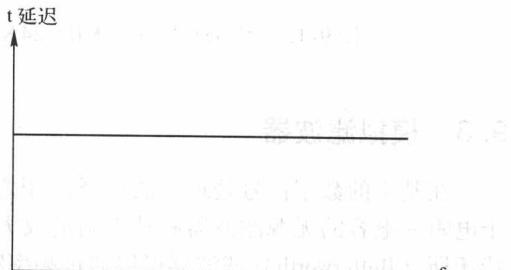


图 9-16 线性相位延时

实际上，在设计滤波器时，线性相位只有在滤波器的通带和过渡带才比较重要，因为其他频率都会被衰减。同样，绝大多数设计采用一种折中的方法，用一个比较小的相位响应变化来赢得相对好的性能；当然，其他性能指标也必须在可接受范围内。

### 9.3.2 时域响应

滤波器也可以用时域响应进行描述。图 9-17 所展示的就是一个低通滤波器在其输入端施加阶跃信号时的典型响应。

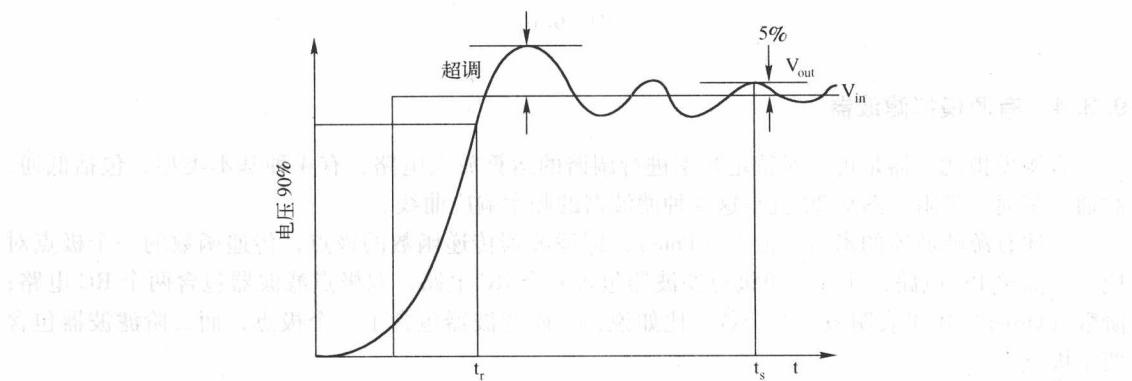


图 9-17 滤波器阶跃响应

在时域内，针对滤波器响应的一些常用性能测试主要有如下几项。

上升时间	输出到达最大值的 90% 所用的时间
建立时间	终值达到 5% 范围内的时间
超调量	经过转换电路后，输出超过期望值的瞬时最大量
振铃现象	终值的震荡

### 9.3.3 模拟低通滤波器

图 9-18 给出了一个简单的模拟低通滤波器，它可用于比较有限冲激响应（Finite Impulse Response, FIR）滤波器和无限冲激响应（Infinite Impulse Response, IIR）滤波器，本章后面将给出二者的相关介绍，另外，也将回顾无源低通滤波器在阶跃信号作用下的工作情况。

时间常数为：

$$T = CR$$

$$T = 10 \times 10^{-9} \times 100 \times 10^3$$

$$T = 10^{-3}$$

$$T = 1 \text{ ms}$$

时间常数是指电容充电达到终值的 63% 所需要的时间。图 9-19a 展示的是滤波器的输入，而图 9-19b 给出了滤波器对于输入信号电平平方函数的平滑效果。

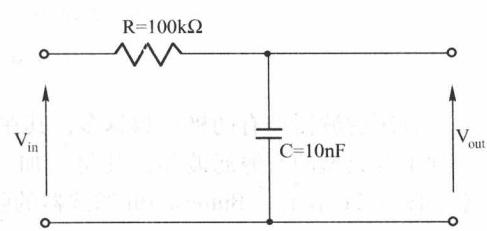


图 9-18 模拟低通 RC 滤波器

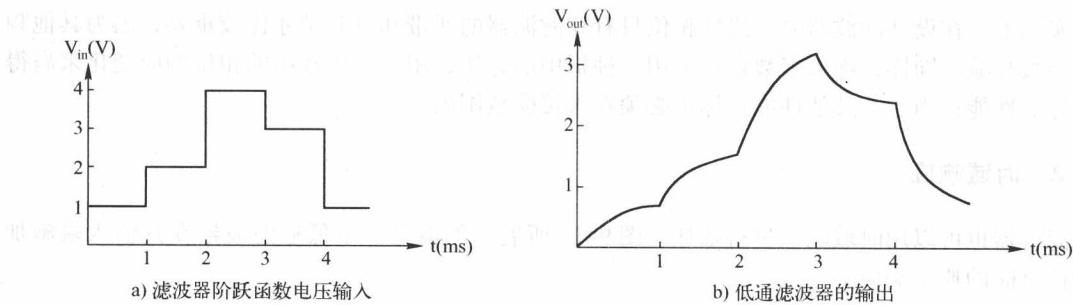


图 9-19

### 9.3.4 有源模拟滤波器

有源模拟滤波器是可以对特定频率进行调谐的运算放大电路，有4种基本类型，包括低通、高通、带通、带阻。图9-20就是这4种滤波器的频率响应曲线。

描述有源滤波器的术语：极点（Pole），即滤波器传递函数的极点，传递函数的一个极点对应一个简单RC电路，比如，单极点滤波器包含一个RC电路，双极点滤波器包含两个RC电路；阶数（Order）用于表明极点的个数，比如说，一阶滤波器包含了一个极点，而二阶滤波器包含两个极点。

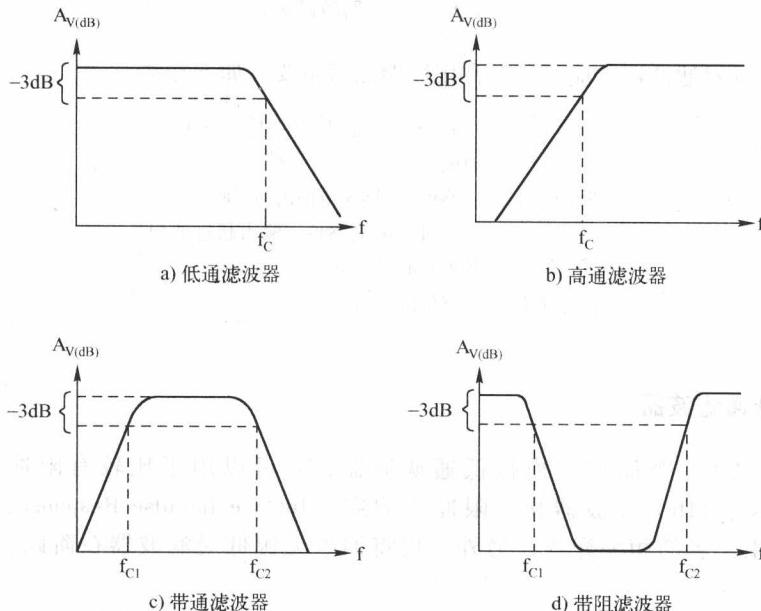


图 9-20 有源滤波器类型

有源滤波器拥有的极点数越多，其在通带之外的增益滚降斜率就越大。Butterworth滤波器是款十分典型的有源滤波器，其每增加一个极点，每十倍频程就会增加一个20 dB的增益滚降率。图9-21给出了Butterworth滤波器的阶数、极点数以及增益滚降之间的关系。

### 9.3.5 有源滤波器的比较

Butterworth滤波器在通过其通带时有一个相对平缓的响应，如图9-22所示。

滤波器类型	极点个数	总的增益下降速度
一阶	1	20dB/decade
二阶	2	40dB/decade
三阶	3	60dB/decade

图 9-21 Butterworth 滤波器

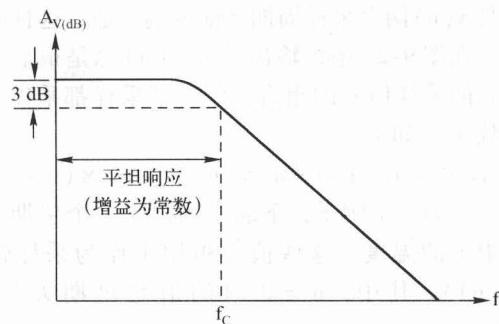


图 9-22 Butterworth 低通滤波器响应曲线

频率刚超过通带时，相对来说，Chebyshev 滤波器的增益滚降斜率比 Butterworth 滤波器的要高（每一极点）。当工作频率离开通带到较远的地方时，这两种滤波器的增益滚降斜率趋于相同。Chebyshev 滤波器的增益在通带范围内是变化的。图 9-23a 显示的是一阶 Chebyshev 滤波器的响应曲线，中带增益的最大值与最小值之差为波动幅度。图 9-23b 给出了 Butterworth 和 Chebychev 滤波器的比较。

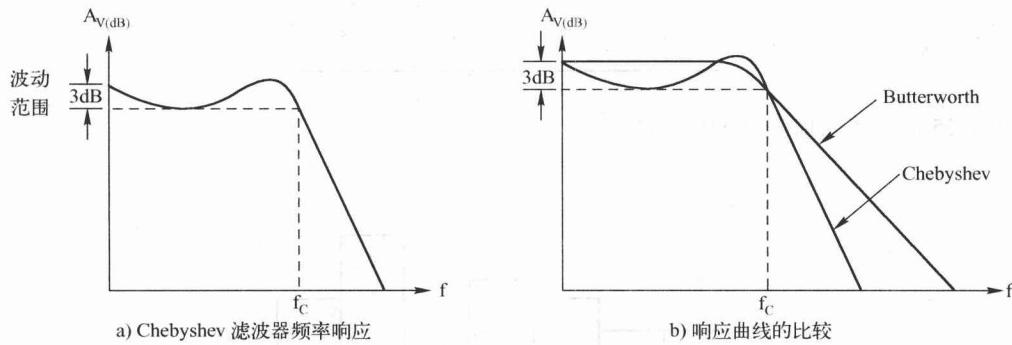


图 9-23

## 9.4 数字滤波器

数字滤波器是 DSP 最普通的应用之一。数字化设计使得设计可以重复使用，并保持完全相同的特性。对于滤波器的数字化来说，还有两个重要的优点。第一，利用该方法可以重新对 DSP 进行编程，从而能够大幅改变滤波器的增益或调整相位响应。例如：系统可以在不改变现有硬件设施的情况下，把一个低通滤波器改变成高通滤波器。第二，利用该方法能够在程序运行的情况下升级滤波器的系数，即“自适应”滤波器。

基本的数字滤波器形式有两种，即有限脉冲响应 (FIR) 和无限脉冲响应 (IIR)，初始的描述都是基于低通滤波器的，但可以转换成其他形式，例如带通或高通。

### 9.4.1 FIR 滤波器

图 9-24 展示的是 FIR 滤波器的方框图。输入信号  $x(n)$  是通过对模拟信号采样得到的一系列不连续的值。在  $x(n)$  序列中， $x(0)$  相当于在时间  $t=0$  时的输入值， $x(1)$  相当于在时间  $t=t_s$  时的输入值， $x(2)$  相当于在时间  $t=2t_s$  时的输入值，以此类推。 $t_s$  是取样周期，并且  $t_s = 1/f_s$ 。

如图 9-24 所示， $z$  是指 Z 变换。可以将图中的  $z^{-1}$  看成是延时一个采样周期  $t_s$  的模块，也称为单位延迟。由此可以看到， $x(n-1)$  的值就是  $x(n)$  一个采样周期之前的输入，同样的， $x(n-2)$

就是  $x(n)$  两个采样周期之前的输入值。这种简单的流程使得 FIR 滤波器更容易理解。

在图 9-24 中，输出信号  $y(n)$  总是最后三个输入的采样信号的组合，每一个采样都乘以一个系数  $a_r$ ，如下：

$$y(n) = a_0 \times (n) + a_1 \times (n - 1) + a_2 \times (n - 2)$$

作为一个例子，下表中列出了一个星期中每天中午的温度。这些值会被用来作为采样输入 ( $x(n)$ )，其中， $n = 0$  时的值是星期天的值，如下：

星期几	周期	$x(n)$	温度 (°F)
星期天	0	$x(0)$	70
星期一	1	$x(1)$	70
星期二	2	$x(2)$	75
星期三	3	$x(3)$	78
星期四	4	$x(4)$	65
星期五	5	$x(5)$	85
星期六	6	$x(6)$	75

图 9-25 展示了对应温度的柱状图。

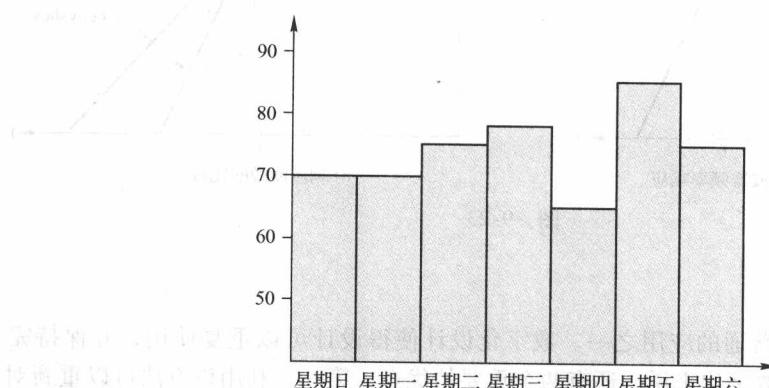


图 9-25 一周温度的柱状图

#### 9.4.2 FIR 滤波器的实现

针对这个例子，我们假设 FIR 滤波器的参数值如下：

$a_n$	Value
$a_0$	0.25
$a_1$	0.50
$a_2$	0.25

FIR 滤波器的实现就变成了如图 9-26 所示的形式。

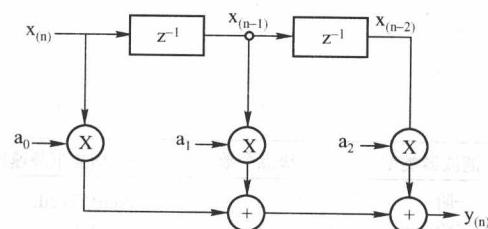


图 9-24 FIR 滤波器框图

根据前面的表格，可以得到  $x(n)$ 、 $x(n-1)$  和  $x(n-2)$  在  $n=0$  时的值为：

$$x(0) = 70$$

$$x(-1) = 0$$

$$x(-2) = 0$$

把上述值带入前面公式，通过乘法与加法运算，得到：

$$\begin{aligned}y(0) &= 0.25 \times 70 + 0.50 \times 0 + 0.25 \times 0 \\&= 17.5\end{aligned}$$

在下一个采样周期，除了将日期偏移一天以外，只需重复上述计算。这样， $x(0)$  就成了星期一；星期天成了  $x(n-1)$  或上一采样周期的值，即：

$$\begin{aligned}y(0) &= 0.25 \times 0 + 0.50 \times 70 + 0.25 \times 70 \\&= 52.5\end{aligned}$$

三天之后，方程就会有三次采样值，结果如下：

$$\begin{aligned}y(0) &= 0.25 \times 75 + 0.50 \times 70 + 0.25 \times 70 \\&= 77.25\end{aligned}$$

在下一个采样周期，星期天的温度值已经退出了方程计算。这反映出了一个事实，就是滤波器限制在三个周期或三个点上进行计算，且有：

$$\begin{aligned}y(0) &= 0.25 \times 78 + 0.50 \times 75 + 0.25 \times 70 \\&= 74.5\end{aligned}$$

图 9-27 给出了滤波器的输出曲线，并假定它已经通过了一个 DAC，在转换点处温度值与滤波器输出一致。

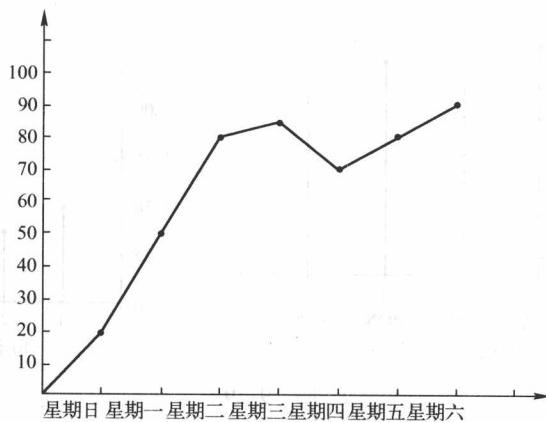


图 9-27 DAC 滤波器的输出

因此可以看出，每天温度变化的平均值与通过模拟低通滤波器输出的值很相近，尽管如此，这里还是以数字方式来实现。

从这个例子中我们也可以看出为什么称之为有限脉冲响应滤波器：计算用了 3 个节拍，并且计算基于  $y(0)$ ，只用到了当前的采样值( $n$ )和之前的两个采样值( $n-1$ )和( $n-2$ )，计算仅使用了有限的采样次数。

### 9.4.3 卷积

正如先前展示的，FIR 滤波器是一个抽头(tap)数为 3 的基本低通滤波器，转换方程为：

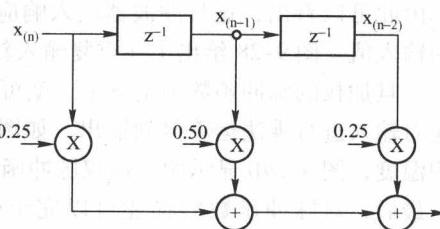


图 9-26 FIR 滤波器

$$y(n) = a_0x(n) + a_1x(n-1) + a_2x(n-2)$$

由此可以看出，这是滤波器输入响应的视觉加权，给出了脉冲函数在  $t=0$  时刻作用于滤波器的输入值。图 9-28 给出了在重复输入新值的情况下产生的脉冲序列。

一旦加权的脉冲函数确定下来，就可以通过滤波器的阶跃响应对输入进行乘法运算得到输出。如图 9-29a 所示是一天的卷积温度，图 9-29b 显示的是加权脉冲函数。

最后，对脉冲函数的描述可以完全确定下来。其初始方程为：

$$y(n) = a_0x(n) + a_1x(n-1) + a_2x(n-2)$$

单位延迟定义为  $z^{-1}$ ，这样一来，输入延时采样就可以定义如下：

$$x(n-1) = x(n)(z^{-1})$$

$$x(n-2) = x(n)(z^{-1})(z^{-1})$$

$$= x(n)(z^{-2})$$

可得 FIR 滤波器的描述方程式如下：

$$y(n) = (a_0 + a_1z^{-1} + a_2z^{-2})x(n)$$

也可以写成：

$$H(n) = \frac{y(n)}{x(n)} = a_0 + a_1z^{-1} + a_2z^{-2}$$

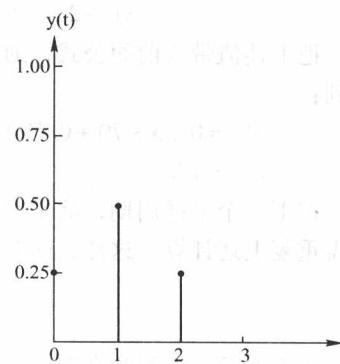
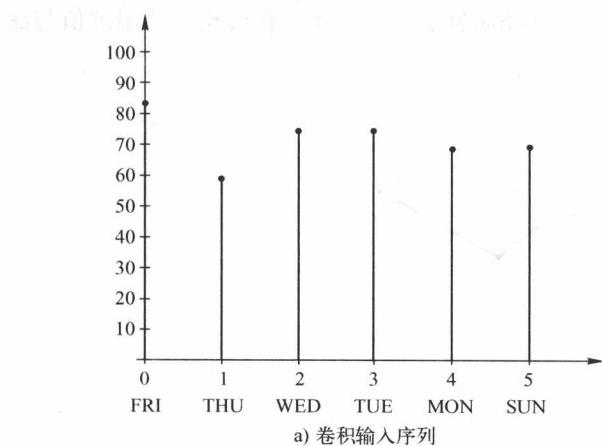
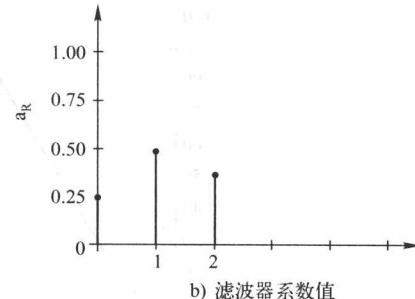


图 9-28 单位脉冲输入的滤波器输出



a) 卷积输入序列



b) 滤波器系数值

图 9-29

传输函数  $H(n)$  是数字滤波器脉冲响应的数学描述。这个方程具有一个很重要的特点，即如果  $z^{-n}$  存在（ $n$  是任意值），则滤波器是内在稳定的。这意味着，与典型的模拟滤波器不同，FIR 滤波器将始终呈现滤波器的特性，而不会产生振荡。

#### 9.4.4 IIR 滤波器

IIR 滤波器是两种基本形式的数字滤波器之一，图 9-30 给出了 IIR 的一个简单实例。

与 FIR 滤波器表示法相似，IIR 滤波器可以表示为：

$$y(n) = x(n) + a_1y(n-1) + a_2y(n-2)$$

$$= x(n) + [a_1z^{-1} + a_2z^{-2}] \cdot y(n)$$

$$= x(n) \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

很明显，从第一个方程中可以发现一点，那就是第二项和第三项都基于  $y(n-t)$ ，换句话说，它们是递归的，使得方程成为无限的。

传递函数的计算式如下：

$$H(n) = \frac{y(n)}{x(n)} = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

从前面的方程可以看出，每一个输出  $y(n)$  都依赖于输入值  $x(n)$  和先前的两个输出值  $y(n-1)$  和  $y(n-2)$ 。一开始，还没有先前的输入采样，并且  $n=0$  时，有：

$$y(0) = x(0)$$

在下一个采样周期有：

$$\begin{aligned} y(1) &= x(1) + a_1 y(0) \\ &= x(1) + a_1 x(0) \end{aligned}$$

接下来  $n=2$  时有：

$$\begin{aligned} y(2) &= x(2) + a_1 y(1) + a_2 y(0) \\ &= x(2) + a_1 [x(1) + a_1 x(0)] + a_2 x(0) \end{aligned}$$

当  $n=3$  时等式就完整了，为：

$$\begin{aligned} y(3) &= x(3) + a_1 y(2) + a_2 y(1) \\ &= x(3) + a_1 [x(2) + a_1 [x(1) + a_1 x(0)]] + a_2 x(0) + \\ &\quad a_2 [x(1) + a_1 x(0)] \end{aligned}$$

从上面的式子可以看出，输出依赖于之前的全部输入。方程会越来越长直到无尽，也因此称之为无限脉冲响应滤波器。

IIR 存在一个问题是可能不稳定。这种滤波器实际上是一系列的反馈回路，在一定的条件下，反馈回路会不稳定。鉴于此，要非常小心地设计以确保得到一个稳定的滤波器。

## 9.5 信号变换

DSP 处理可以采用 FIR 滤波器和 IIR 滤波器实现。信号变换的另一种方法是把信号从时域变换到频域，频谱分析仪就是一个典型的例子，其他例子包括通信信道带宽分析和数字语音识别等，这三个应用都需要将时变信号变换到频域。

### 9.5.1 相量模型

相量模型是描述信号的一种基本方法。相量是一个旋转复平面矢量，用大小  $A$  和角转速  $\omega$  (radian/sec) 表示，如图 9-31 所示。

相量模型可以扩展到离散时间，信号只出现在由采样周期  $T_s$  确定的特定点上。用一个离散变化的  $n$  取代连续变化的时间  $t$ ，这样相量就按  $T_s$  推进。

余弦信号可以用一对共轭相量表示，这意味着它们有相同的实部值 ( $a$ ) 和相反的虚部值 ( $b$ )。从这里我们可以得出一个有趣的事，就是所有的实信号都必须由一对共轭相量组成，这样矢量的求和总会落在实轴上。

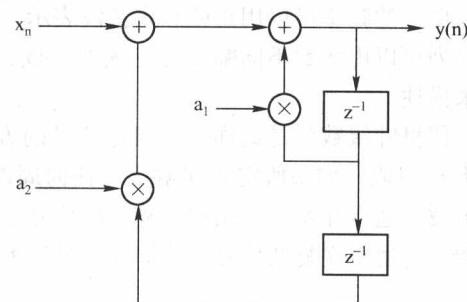


图 9-30 IIR 滤波器

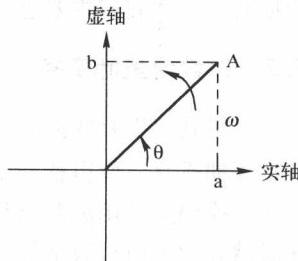


图 9-31 相量模型

### 9.5.2 傅里叶级数

单一的频率可以用正弦和余弦波表示，复杂的波形也可以用它们来表示。例如，一个矩形脉冲序列可以由无数不同幅度的正弦波组成，同样，一个复杂的周期性信号也可以用许多向量之和来描述。

傅里叶级数就是这样一种描述信号的方法。这种方法假定相量集表示的频率由一些基本的频率  $f_0$  组成。如果假定  $N$  足够大，任何周期信号都可以由傅里叶级数表示，单个的频率部分称为谐波。通常在实际应用中，复杂的信号是非周期性的。使用谐波不相关的相量可以产生傅里叶级数，这通常在复杂信号是非周期信号（实际应用中，大部分情况都是这样）的情况下适用：

$$x(t) = \sum_{k=-N}^N C_k e^{j(\omega_k t)}$$

任意波形都可以利用上式所示的傅里叶级数来表示。

### 9.5.3 离散傅里叶级数

数字信号处理应用于数字领域，连续的方程需要转换为离散的或数字的领域，以便一些有用的公式可以应用。这时需要做的就是用以  $\omega_0 T_s$  为步长的抽样取代连续的函数  $t$ 。

离散信号的频率响应是以  $T_s$  为周期的，这是数字信号处理很重要的一个特征。现在，向量模型已用于描述一些离散信号。

### 9.5.4 傅里叶变换

实际应用中，信号在本质上很少是周期性的。傅里叶级数需要修改并改进，以适应这一事实。这可以通过整合来解决。给定一个通用的傅里叶级数，其所有频率都是谐波相关的，如下：

$$\omega_k = K\omega_0$$

最终信号不是周期性的信号，可以通过下式表示：

$$\omega_0 \rightarrow 0$$

这个方程告诉我们，独立相量的频率没有“最小公分母”，相量的个数趋于无穷大，它们的和成为一个整体。

$$X(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} X(\omega) \cdot e^{j(\omega t)} \cdot d\omega$$

在前面的方程中，假定信号的振幅是一个频率函数，即  $X(\omega)$ ， $X(t)$  的逆方程可以用来定义  $X(\omega)$ ，如下：

$$X(\omega) = \int_{-\infty}^{+\infty} X(t) \cdot e^{j(-\omega t)} \cdot dt$$

这个方程允许在频域内对连续信号的幅频响应进行计算。这两个方程和在一起被称为傅里叶变换对。虽然它们主要进行数学计算，但不能直接用于 DSP，因为还需要一个离散形式。

### 9.5.5 离散傅里叶变换

为了得到傅里叶变换的离散方程，连续时间变量  $t$  必须用离散变量  $nT_s$  代替。在频谱  $\pm \pi/T_s$  以外按周期重复，使得整合变量  $\omega T_s$  为一个整数。

逆变换仍然采用离散求和的方式，主要是因为  $x(n)$  只在  $nT_s$  时刻才有效。这样一对方程允许在时域和频域内对数字信号进行变换。

在这一点上，有一种描述离散时变信号的方法，称作向量模型，该模型已经发展到可以用来

表示傅里叶级数；更进一步说，已经被用来计算 DFT 变换对。现在已经存在这样一对方程，它们能够提供使任意离散信号在时域和频域内进行变换的方法。

关于离散 DFT 的  $X(\omega)$  还存在一个问题，在现实世界中无法实现无限求和，在实际项目中对每一个频率能够计算的数目是有限制的。通过限定采样  $x(n)$  可以很简单地解决无限问题，这种方法称作“窗函数”，频谱结果可以由下面的方程给出：

$$X_N(\omega) = X(\omega) * W(\omega)$$

其中， $X_N(\omega)$  表示用采样次数  $N$  表述的频谱函数。 $W(\omega)$  代表了窗口的频谱， $*$  代表  $W(\omega)$  和  $X(\omega)$  卷积，和第 8 章提到的卷积概念一样，因此这个方程可以改写如下：

$$X_N(\omega) = \sum_{r=0}^{N-1} X_r \cdot \omega(N-r)$$

现在假设已经选定了一个窗函数，我们必须知道如果只应用有限数目的频率的影响，需要多少频率才能确保其准确度。但这没有一个明确的答案，大体上说，最适宜的相量数量等于在  $x(n)$  中原始点的个数。最简单的理解是，假定  $x(n)$  的部分窗口有一个长序列的周期（见图 9-32），其周期为  $NT_s$ ，频率为  $\omega_s/N$ ，如果这能够实现，该序列就可以按照傅里叶级数来对待，并且可以看出，其频谱中包含了  $N$  个相量，如图 9-33 所示。

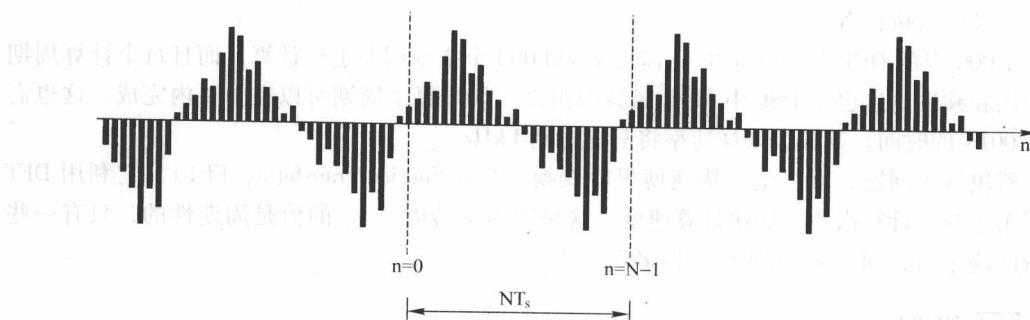


图 9-32 DFT 窗口

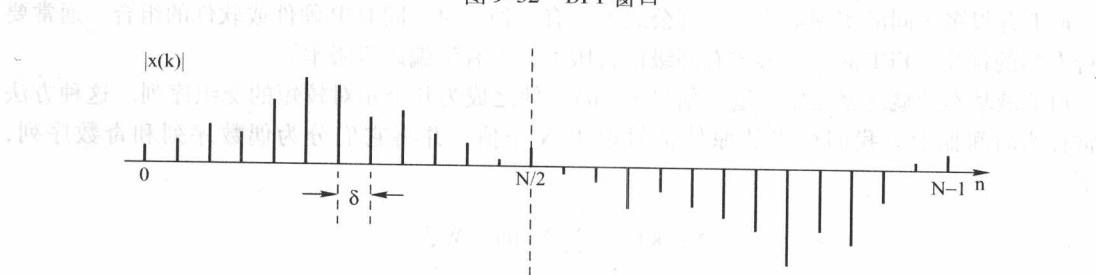


图 9-33 相量间距

在 DFT 公式的实际应用中，还有一个参数称为旋转因子，可以作为取代参数。该参数可以由以周期  $\omega_s$  重复的 DFT 频谱衍生出来，即：

$$N\delta = \omega_s$$

接下来将频率标度数字化，这样就可以用  $k$  取代  $\omega$  重写频谱：

$$X(k\delta) = K(\delta) = \sum_{n=0}^{N-1} x(n) \cdot e^{j(-k\delta T_s n)}$$

其中：

$$\omega_s = \frac{2\pi}{T_s}, \quad \delta = \frac{\omega_s}{N}$$

则频谱方程可以重写如下：

$$X_N(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\left(\frac{2\pi kn}{N}\right)}$$

这个就是广泛应用的实际 DFT 公式。

旋转因子用  $W_N$  来表示，其定义为：

$$W_N = e^{-j\left(\frac{2\pi}{N}\right)}$$

将 DFT 变换对公式以最常用形式重写如下：

$$X_N(k) = \sum_{n=0}^{N-1} X(n) \cdot W_N^{kn}$$

$$X(n) = \frac{1}{N} \sum_{k=0}^{N-1} X_N(k) \cdot W_N^{-kn}$$

## 9.6 快速傅里叶变换

DFT 方程的离散形式已经推导出来，但必须面对实际计算时间的问题。从方程中可以看到，有重复的加法和乘法系列计算。这是因为指数存在  $k$  和  $n$ ，必须通过  $N$  值生成整个范围的输出相量，因此，必然会执行  $N^2$  次计算。

一个 1 000 点的 DFT ( $N = 1 000$ )，需要 1 000 000 个指令周期进行计算，而且每个计算周期都需要加法和乘法。这也是 DSP 中 MAC 函数的由来。如果每个周期可以在 1 ns 内完成，这也需要花费 0.001 s 的时间，其最大采样速率将只能达到 1 kHz。

为了解决这个问题，就产生了快速傅里叶变换 (Fast Fourier Transform, FFT)。它利用 DFT 的内在冗余来减少计算次数，加快计算速度。这是因为旋转因子  $w_n$  的值是周期性的，只有一些有限数量的确定值，同一  $w_n$  值要计算很多次。

### 9.6.1 FFT 的执行

FFT 有很多不同的实现形式，实现公式不只有一种，对不同 DSP 硬件或软件的组合，通常要进行专门的优化。FFT 的实现形式有高级语言版本，也有汇编语言版本。

FFT 最基本的概念就是精简输入信号  $x(n)$ ，使之成为几个相对较短的交织序列。这种方法通常称为时间抽取。我们可以从原始信号取出  $N$  个值，并将它们分为偶数序列和奇数序列，如下：

$$X_N(k) = \sum_{n=0}^{N-1} X(n) \cdot W_N^{kn}$$

经过适当的替代后，可以得到：

$$X_N(k) = \sum_{r=0}^{\frac{N}{2}-1} x(Zr) \cdot (W_N^z)^k + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(Zr+1) \cdot (W_N^z)^{k+1}$$

应用  $W_N$  的最初定义后，有：

$$X_N(k) = \sum_{r=0}^{\frac{N}{2}-1} x(Zr) \cdot W_N^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(Zr+1) \cdot W_N^{r+k}$$

这个式子也可以写成：

$$X_N(k) = G(K) + W_N^K H(K)$$

$G(K)$  是 DFT 的偶数点部分，而  $H(K)$  是奇数点部分。可以注意到，初始 DFT 转换成了两个长度为  $N/2$  的小一些的 DFT。还是计算  $N = 1 000$  的情况，现在需要  $500^2 + 500^2 + 500 = 500 500$

条指令周期，比一开始的 1 000 000 少了  $\frac{1}{2}$ 。

如果采样数 N 是 2 的幂，指令数可以进一步减少，将 N 分半处理。事实上，这个过程可以一直重复下去，直到只有两点 DFT。这是最常用的 FFT 变换，称之为基 -2 变换 (radix -2)。例如，如果 N = 8，x (n) 的定义如此下：

$$n = \{0, 1, 2, 3, 4, 5, 6, 7\}$$

接着，可以抽取为两个序列：

$$n = \{0, 2, 4, 6\} \text{ 和 } \{1, 3, 5, 7\}$$

然后，再次抽取得四个序列：

$$n = \{0, 4\}、\{2, 6\}、\{1, 5\} \text{ 和 } \{3, 7\}$$

至此，FFT 可以执行四次两点 DFT 计算得到，这是最简单的计算。尽管如此，旋转因子提升了每次序列抽取的时间，而且每次计算旋转因子都不同。FFT 算法必须能够在不使计算太复杂的情况下处理这些因子。

于是 FFT 可写作：

$$X_4(k) = [x(0) + x(2) \cdot W_4^{2k}] + W_4^k [x(1) + x(3) \cdot W_4^{2k}]$$

k 取所有值的情况如下：

$$X_4(0) = [x(0) + x(2) \cdot W_4^0] + W_4^0 [x(1) + x(3) \cdot W_4^0]$$

$$X_4(1) = [x(0) + x(2) \cdot W_4^2] + W_4^1 [x(1) + x(3) \cdot W_4^2]$$

$$X_4(2) = [x(0) + x(2) \cdot W_4^0] + W_4^2 [x(1) + x(3) \cdot W_4^0]$$

$$X_4(3) = [x(0) + x(2) \cdot W_4^2] + W_4^3 [x(1) + x(3) \cdot W_4^2]$$

## 9.6.2 DFT 蝶形变换

图 9-34 显示了 4 点 DFT 变换计算的信号流程图，利用了经典的蝶形变换。圆圈内的数字表示  $W_4$  的幂，需要在该阶段进行计算。

图 9-34 中图形的每一个部分由一定数量的蝶形组成。当倍增因子为  $x=0$  和  $y=-1$  时，便是一种被广泛接受的蝶形变换形式如图 9-35 所示。

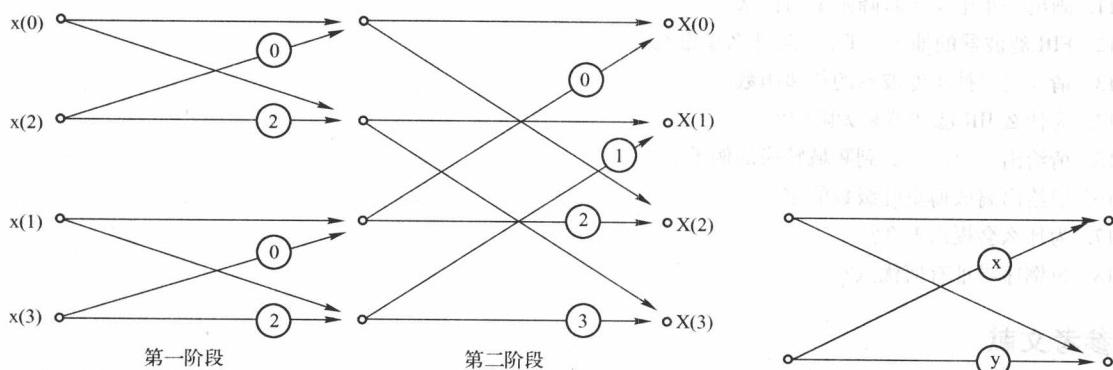


图 9-34 4 点蝶形变换

图 9-35 FFT 蝶形变换

## 9.7 表寻址

FFT 的实现要用到两个表，一个是输入值列表，另一个是相应的输出列表。对于 4 点 DFT，这不是问题，但当点数为 1 024 或更大时，这就很复杂了。

大部分通用 DSP 都有位倒序寻址能力。位倒序寻址是一种寻址模式，输入值按升序存储，处理器以正确的顺序选择这些数据。例如，对于一个 8 点 DFT，这种定址模式把输入值按升序排列，程序按正序寻找他们：

```

start at 000 = x(0)
000 + 100 = 100 = x(4)
000 + 100 = 100 = x(4)
100 + 100 = 010 = x(2)
010 + 100 = 110 = x(6)
110 + 100 = 001 = x(1)
001 + 100 = 101 = x(5)
101 + 100 = 011 = x(3)
011 + 100 = 111 = x(7)

```

现在很多新的高性能 DSP 都有独立的地址生成单元，可以消除任何访问输入数据表的延时，大大提高 DSP 的整体性能。

## 习题

1. 请给出 DSP 的定义。
2. 请用图表说明 DSP 处理流程的 5 个步骤。
3. 在 DSP 系统中为什么要应用换能器？
4. 什么样的处理器体系结构被用来实现 DSP，为什么？
5. MAC 函数模块有什么作用？
6. 通带和阻带有什么区别？
7. 为什么说信号的相位很重要？
8. 画图说明典型的低通滤波器的输出波形。
9. 列举 4 种典型的有源滤波器类型。
10. 4 阶滤波器有多少个 RC 电路？
11. 画出一年中每个月降雨量的柱状图。
12. FIR 滤波器的抽头 (Tap) 是什么意思？
13. 请写出三抽头滤波器的传递函数。
14. 为什么 IIR 滤波器是无限的？
15. 请给出一个从时域到频域转换的例子。
16. 请给出离散傅里叶级数的定义。
17. 为什么会提出 FFT？
18. 位倒序寻址有何优点？

## 参考文献

- ADSP-BF533 Blackfin® processor hardware reference revision 3.1. May 2005. Part Number 82-002005-01. Analog Devices.
- ARM1022E technical reference manual, revision: r0p2. 2005. ARM Limited.
- Francis, Hedley. 2001, May. ARM DSP-enhanced extensions, ARM White Paper. ARM Ltd.
- Grover, Dale & John R. Deller. 1999. *Digital signal processing and the microcontroller*, Upper Saddle River, NJ: Prentice Hall.
- Marven, Craig & Gillian Ewers. 1996. *A simple approach to digital signal processing*. New

- York: John Wiley & Sons, Inc.
- MC9S12DT128 device user guide 9S12DT128DGV2/D, V02.15 05. October 2005. Motorola, Inc.
- MIPS32 4KE processor core family software user's manual, Document Number: MD00103, Revision 01.08. January 30, 2002. MIPS Technologies.
- Smith, Steven W. 1999. *The scientist and engineer's guide to digital signal processing*, San Diego, CA: California Technical Publishing.
- TMS320C55x DSP, programmer's guide, preliminary draft, TI Literature Number: SPRU376A. August 2001.
- Xtensa architecture and performance: October 2005. White Paper. Tensilica, Inc.

## 第 10 章 |

Embedded Microcontrollers and Processor Design

# 模糊逻辑

- 本章目标：介绍模糊逻辑概念在基于嵌入式控制器的系统中的应用

- 主要内容：

1. 模糊逻辑的概念
2. 模糊逻辑的数学基础
3. 模糊逻辑的应用

## 10.0 模糊逻辑

通常，我们认为测量结果在表达方面是相对的。外界温度可以是冷、热或者温和，相应的说法取决于当前的测量结果。例如，温和的温度随我们所处季节的不同而变化，菲尼克斯温度温和的时候在斯德哥尔摩就不一定温和了。

模糊逻辑是处理相对结果的方法学。伯克利大学的罗特夫·扎德博士于 1962 年首次提出模糊逻辑的概念，他为处理现实世界中事物的相对性建立了数学基础。模糊逻辑能够用于状态控制，是实现控制算法的一种方式；与典型的二进制开关型算法相比，它能够更快地收敛于期望值。图 10-1 给出了一个微波炉控制器控制逻辑的例子。

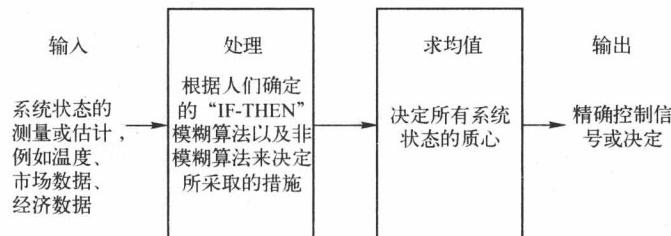


图 10-1 模糊逻辑控制 - 分析方法 (Freescale 2007 版权许可)

模糊逻辑被认为是一种用于数据分类和处理的较好算法，但是由于它模仿人类的控制逻辑，已经被证实为许多控制系统的极佳选择。从小型的手持式产品，到大型的计算机控制系统，模糊逻辑几乎可以应用到任何系统中。它使用一种不精确但描述能力很强的语言来处理输入数据，更像一个人工操作员。模糊逻辑操作器具有较强的鲁棒性，调节范围宽广，除了首次实现时有一点调节之外，数据输入及工作通常都十分正常。

为了得到温水，二进制方法通过将冷水阀或热水阀全开或全关来进行调节，模糊逻辑与其不同，它试着通过部分转动两个阀门来产生温水。结果，模糊逻辑控制水温更为精确，也能使水更快地达到期望的温度。事实上，采用二进制控制方法实现混合水温时，不可能达到精确的容许范围。

模糊逻辑提供了一种收敛于终值的方法，该方法没有二进制算法中在终值附近的波动。事实上，二进制算法不能达到期望值，而且会在可接受的误差范围内波动（见图 10-2a）。模糊逻辑提供了一个更快更精确的收敛值（见图 10-2b）。随着微控制器性能的提高，模糊逻辑电路可以成为 CPU 逻辑功能的一部分，这在 Freescale HCS12X 微控制器中已有所体现（见图 10-3）。

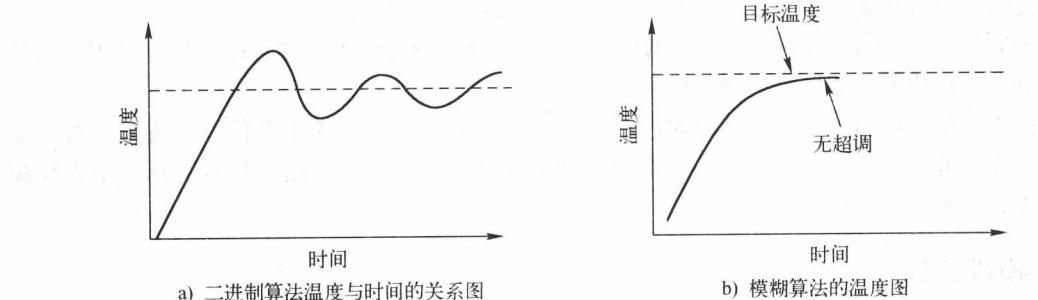


图 10-2

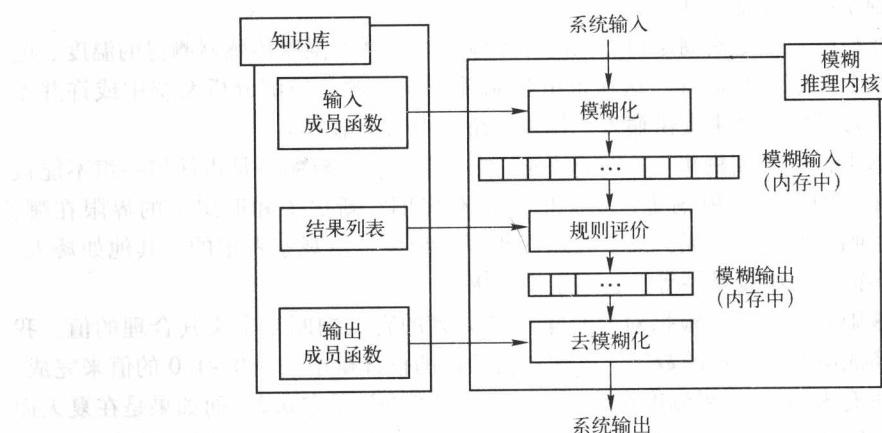


图 10-3 Freescale S12XCPU 模糊逻辑功能 (Freescale 2007 版权许可)

## 10.1 模糊逻辑方法

我们采集的很多信息都不能给予精确的定义，例如加快车速，这就叫做模糊输入。然而，一些“输入”相当精确而不模糊，比如速度计的读数。所有这些信息的处理都不能精确地定义，就叫做模糊处理，模糊逻辑理论家会称之为模糊算法调用。

模糊逻辑与人类大脑的功能很相似，而且大脑的功能可以被模仿，因此机器可以执行一些类似大脑的功能。在这点上不要与人工智能相混淆，人工智能想使机器的功能像人类一样精确（或者表现更佳）。模糊逻辑控制和分析系统在本质上是电动机械的，或者仅与在所有情况下由人类语言中的“IF-THEN”规则声明的数据有关，比如经济数据。

模糊逻辑方法是分析和控制的结合，如图 10-4 所示。它由如下 3 个基本部分组成：

输入	处理	求均值	输出
系统状态的测量或估计，例如温度、市场数据、经济数据	根据人们确定的“IF-THEN”模糊算法以及非模糊算法来决定所采取的措施	决定所有系统条件的质心	精确的控制信号或决定

图 10-4 模糊逻辑控制分析方法

- 接收待分析系统的一个或多个测量结果或者系统状态的估计。
- 根据基于人类的、模糊的“IF – THEN”规则来处理所有输入，这些规则都可以用普通的语言以及传统的非模糊处理过程来表达。
- 将所有单个规则中的输出结果通过求均值和加权使最后输出为单个信号或决策，这个信号或决策决定做什么或者告诉一个受控制系统做什么。最终，输出信号达到一个去模糊化的精确值（crisp value）。

## 10.2 模糊感知

模糊感知就是对没有经过精确测量的物理量进行估计，并对该物理量赋予一个直觉值。事实上，模糊逻辑认为基本上所有事物都有一点模糊，不论测量仪器多么精确。模糊感知在处理和分析一个模糊逻辑控制系统时起基础性作用。

已测量的数据、非模糊数据是模糊逻辑方法的原始输入，包括由温度传感器测得的温度、电机速度、经济数据以及金融市场数据等。这在机电控制系统或金融与经济分析系统中或许并不常见，但是人们可以通过模糊感知来提供输入，作为回路当中的一部分。

在模糊逻辑的文献中，有“模糊集”（Fuzzy Set）的概念。一个模糊集是指任何一组不能被精确定义的事物。例如“旧房子”模糊集，一个旧房子有多旧？新房子和旧房子的界限在哪？十五年的房子是旧房子吗？四十年的呢？三十九点九年呢？结果是由观察者定的。其他如矮人、热天、高压气体、一小群人和中度黏性等，也属于模糊集。

当人类是分析的基础时，有必要根据对模糊集中各元素的直觉判断来定义其合理的值。我们必须把人类模糊转换成电脑可使用的数字，可通过给条件的估计赋予一个0~1.0的值来完成。例如，对“这个房子里有多热”，如果温度在0°C以下，人们可以赋予它0.2，而如果是在夏天而且天比较热，同时未开空调，人们可以定义这个房间为0.9甚至1.0。这些认识都是模糊的，只是直觉的判断，不是精确的测量结果。

通过做模糊估计，用0来表示最低水平，用1来表示最高水平，进而建立基本的模糊逻辑方法分析规则。对于复杂系统或只能依据人类经验才能操作的系统而言，这样做的输出结果看起来不错，起码比什么都不做要好得多。如果不是模糊规则，我们可能真的什么都做不了。

模糊逻辑利用了人们的常识。这些常识，对于新系统而言，来源于那些看上去合理的东西；对于以前有过人工操作的系统来说，则来源于经验。模糊逻辑分析和控制的对象可以包括物理控制，例如机器速度控制或水泥厂控制、金融和经济决策、心理条件、生理条件、安全条件以及保安条件等。

## 10.3 模糊逻辑的术语

下面对模糊逻辑中用到的一些术语进行解释，这些术语最初是由Dr. Zadeh在提出模糊逻辑的概念时提出的。

**模糊**——一个系统分析规则的模糊程度可以在非常精确（这种情况下不能称为“模糊”）和“模糊”（基于人的主观认识）之间变化。因此，模糊还是非模糊，与一个系统分析规则的精确度有关。

一个系统的分析规则不需要以人的模糊感知为基础。例如，有如下规则，“如果压力传感器测到锅炉的压力上升到较危险的值600Psi时，把一切设备都关掉”，这个规则就不“模糊”。

**不相容原理**——随着系统复杂性的增加，越来越难于甚至于不可能对系统的行为做出精确估计，更甚至于达到一定的复杂程度，只能使用源于人们经验的模糊逻辑方法作为解决问题的唯一途径。

**模糊集**——模糊集可以是任何可以用语言描述的条件，如矮人、高个子妇女、热、冷和新建筑等，这些条件可以被赋予一个0~1.0之间的值。例如，一个妇女六英尺三英寸高，一般地，

根据大多数人的标准，可以说她高，并赋予她0.98的身高分级。依据这条推理路线，可以无穷地向下推理，对许多事物给出0~1之间的评级。

**隶属度**——隶属度是一个模糊集中的条件在0~1之间转换的定位。如果一个特定建筑在新建筑中新的程度的定位是0.7，那么该建筑在新建筑中的隶属度就是0.7。

在模糊逻辑方法控制系统中，隶属度是按照以下方式使用的。例如，测量速度时会出现“太快”的隶属度0.6以及“不需要变速”的隶属度0.2，系统程序会在“太快”和“不需要改变”之间计算质心，以决定反馈措施作为控制系统的输入。

**信息总结**——人们对信息的处理过程不是基于二值逻辑、开关逻辑或者二选一逻辑，而是基于模糊感知、模糊事实和模糊推理等，所有这些会产生一个均值的、总结性的、归一化的输出，人们把这个输出看作是精确的决策值，先用语言进行描述，然后记下来，最后按照它来采取措施。这样做也是模糊逻辑控制系统的目

系统可能有大量的输入数据，但是我们有能力操作模糊集，并归纳产生输出结果。

**模糊变量**——像红色、蓝色等这样的词汇是模糊的，可以有很多暗色和色调，它们只是人们的主观看法，不是基于精确测量得出的，都是模糊变量。例如，如果系统的速度由于模糊估计以及模糊规则产生，则该“速度”便是模糊变量。

**语言变量**——与语言相关的事物，这里是指与普通的语言词汇相关。速度是一个模糊变量，加速设置也是一个模糊变量，语言变量的例子有稍微快的速度、非常高的速度、实在很慢的速度、过高的加速设置及合适的加速设置等。

当用描述性文字来修饰一个模糊变量时，比如有点快、很高、真得很慢等，它就变成了语言变量。语言变量的主要功能是用来处理前面提到的复杂系统，因为它们太复杂而不能通过传统的数学和工程方程来解决。语言变量出现在反馈循环控制系统中，而且在“IF-THEN”条件语句中可以彼此产生关系。例如，如果速度太快，则退回高加速设置。

**论域**——假设考虑的对象为妇女，那么各地的妇女都属妇女的范围，如果选择妇女来讨论，那么各地的妇女都属于我们的论域。

因此，论域是说明一种特定类别范围的所有对象的方式，通常用单个词来命名，这一特定种类正好是有关模糊逻辑方法或配合模糊逻辑方法工作的。论域是由模糊集组成的。妇女的论域由职业女性、高的妇女、矮的妇女、漂亮的妇女等来组成。

**模糊算法**——算法就是程序，比如计算机程序中的指令。模糊算法是一段程序，通常是计算机程序，由与语言变量相关的语句组成。例如：

如果“绿色的X”很大，则让“高的Y”很小。如果蒸汽机锅炉温度的变化率太高，那么就大幅度关小加热器。

## 10.4 模糊专家系统

模糊专家系统是用模糊逻辑取代布尔逻辑的专家系统。换句话说，模糊专家系统就是用来对数据进行推理的隶属函数和规则的集合。与主要用于符号推理的传统专家系统不同，模糊专家系统主要面向数字处理。

模糊逻辑系统中的规则通常类似于如下形式：

If x is low and y is high then z = medium

这里的x和y是输入变量，z是输出变量，low是定义在x上的隶属函数（模糊子集），high是定义在y上的隶属函数，medium是定义在z上的隶属函数。

if和then之间的规则部分是规则的前提条件，是描述该规则可应用程度的模糊逻辑表达，then后面的规则部分是规则的结论，定义了每一个或更多输出变量的隶属函数。大部分用于模糊

专家系统的开发工具都允许在一条规则内部包含多个结论，一个典型的模糊专家系统都有多个规则，这些规则全部集中起来称作规则库或知识库。模糊逻辑规则的标准定义是：

- 1) 非（否定准则）： $\text{truth}(\text{not } x) = 1.0 - \text{truth}(x)$ ；
- 2) 交集（最小值准则）： $\text{truth}(x \text{ and } y) = \min(\text{truth}(x), \text{truth}(y))$ ；
- 3) 并集（最大值准则）： $\text{truth}(x \text{ or } y) = \max(\text{truth}(x), \text{truth}(y))$ 。

#### 10.4.1 推理过程

由于规则和隶属函数已经定义，就可以应用它们由输入变量的特定值来计算输出变量，这个过程就叫做推理。在模糊专家系统中，推理过程包含如下4个子过程：

- 模糊化。
- 推理。
- 合成。
- 去模糊化。

注意：去模糊化过程是可选的。

假设变量  $x, y, z$  在区间  $[0, 10]$  上取值，假设  $t$  是区间  $[0, 10]$  上的变量，且：可以定义下面的隶属函数和规则：

$$\text{Low}(t) = 1 - t/10$$

$$\text{High}(t) = t/10$$

可以定义下面的隶属函数和规则：

- 规则 1：If  $x$  is low and  $y$  is low then  $z$  is high
- 规则 2：If  $x$  is low and  $y$  is high then  $z$  is low
- 规则 3：If  $x$  is high and  $y$  is low then  $z$  is low
- 规则 4：If  $x$  is high and  $y$  is high then  $z$  is high

注意：不是给输出变量  $z$  赋予一个单值，每个规则都赋予  $z$  一个 low 或 high 的完整模糊集。对这个例子，还要注意如下几点。

- 1) 对于所有的  $t$ ,  $\text{low}(t) + \text{high}(t) = 1.0$ 。这一点不是必须的，但通常都是这样。
- 2)  $\text{low}(t)$  取最大值时对应的  $t$  值与  $\text{high}(t)$  取最小值时对应的  $t$  值一样，反之亦然。这也不是必须的，但通常也都是这样。
- 3) 所有的变量都用相同的隶属函数。这也不是必须的。

#### 10.4.2 模糊化

在模糊化的子过程中，输入变量的隶属函数通过输入变量的实际值来确定规则前提条件的真实度。一个规则的前提条件的真实度有时被定义为它的 alpha 值。如果一个规则的前提条件有一个非零的真实度，这个规则就有效。

例如，图 10-5 展示了  $X$  和  $Y$  取值的四种组合，alpha1 对应规则 1 的结果，依此类推。

$x$	$y$	$\text{Low}(x)$	$\text{High}(x)$	$\text{Low}(y)$	$\text{High}(y)$	$\text{alpha1}$	$\text{alpha2}$	$\text{alpha3}$	$\text{alpha4}$
0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
0.0	5.8	1.0	0.0	0.42	0.58	0.42	0.58	0.0	0.0
3.8	6.4	0.62	0.38	0.36	0.64	0.36	0.62	0.36	0.38
7.2	7.1	0.28	0.72	0.29	0.71	0.28	0.28	0.29	0.71

图 10-5 模糊化规则表

### 10.4.3 推理

在推理子过程中，我们要计算每个规则的前提条件的真实度，在每个规则的结论部分都会用到它。这使得一个模糊子集会赋给每个规则的各个输出变量。

基本的推理方法（推理规则）包括 MIN 和 PRODUCT。在 MIN 推理方法中，输出隶属函数被消减至一定大小，这个大小与规则前提的真实度相关，符合模糊逻辑 AND 操作的传统解释。在 PRODUCT 推理方法中，输出隶属函数用计算的规则的前提条件真实度来衡量。

例如图 10-5 所示，规则 1 中  $x=0.0$ ,  $y=5.8$ ，计算出规则前提的真实度是 0.58。对这个规则，MIN 推理方法给  $z$  赋予由隶属函数定义的模糊子集：

$$\text{rule1}(z) = \begin{cases} z/10, & \text{if } z \leq 5.8 \\ 0.58, & \text{if } z \geq 5.8 \end{cases}$$

在相同的情况下，PRODUCT 推理方法会赋给  $z$  一个由隶属函数定义的模糊子集：

$$\begin{aligned} \text{rule1}(z) &= 0.58 * \text{high}(z) \\ &= 0.58 * z/10 \\ &= 0.058 * z \end{aligned}$$

### 10.4.4 合成

在合成子过程中，分配给每一输出变量的所有模糊子集组合在一起，为每个输出变量分别形成一个模糊子集。

合成规则主要有两个，即 MAX 合成和 SUM 合成。在 MAX 合成中，模糊子集的组合输出通过在推理规则赋给各个输出变量的所有模糊子集上逐点求取最大值产生。在 SUM 合成中，模糊子集的组合输出通过在所有规则赋给各个输出变量的推理模糊集上逐点求和产生。需要注意的是，SUM 合成可能会导致真实度大于 1。也正是由于这个原因，SUM 合成只有在其后面使用去模糊化方法（比如重心法）时才可使用，该方法对于先前情况没有影响。

例如，假定  $x=0.0$ ,  $y=5.3$ 。MIN 推理方法会把以下四个模糊子集赋给  $z$ :

$$\begin{aligned} \text{rule1}(z) &= \begin{cases} z/10, & \text{if } z \leq 5.3 \\ 0.53, & \text{if } z \geq 5.3 \end{cases} \\ \text{rule2}(z) &= \begin{cases} 0.37, & \text{if } z \leq 5.3 \\ z/10, & \text{if } z \geq 5.3 \end{cases} \\ \text{rule3}(z) &= 0 \\ \text{rule4}(z) &= 0 \end{aligned}$$

MAX 组合会产生以下模糊子集：

$$\text{fuzzy}(z) = \begin{cases} 0.37, & \text{if } z \leq 3.7 \\ z/10, & \text{if } 3.7 \leq z \leq 5.3 \\ 0.53, & \text{if } z \geq 5.3 \end{cases}$$

PRODUCT 推理会赋给  $z$  以下四个模糊子集：

$$\begin{aligned} \text{rule1}(z) &= 0.053 * z \\ \text{rule2}(z) &= 0.37 - 0.037 * z \\ \text{rule3}(z) &= 0.0 \\ \text{rule4}(z) &= 0.0 \end{aligned}$$

### 10.4.5 去模糊化

实际应用中，有时候需要检查合成过程产生的模糊子集，但是大部分情况下，这些模糊取值需要转化成一个数字，即一个精确值。这就是去模糊化子过程所要做的。

去模糊化的方法有许多，包括重心法（CENTROID）、最大值法（MAXIMUM）、单点法和权值平均法。最常用的两组方法是重心法和最大值法，而最流行的是重心法。在重心法中，可以通过给模糊值找出隶属函数重心的变量值计算输出变量的精确值。

去模糊化的重心法用于计算前面步骤中合成的模糊集的质心。这种方法的数值较为密集，需要对集合整体进行计算。为避免产生无效输出的情形，还需要处理输出集合的重叠现象。

采用最大值法时，在模糊子集中选取最大真实度值的变量值作为输出变量的精确值。最大值法有几种变化，它们只是在出现多个最大真实度值的变量时才有所区别。其中，最大平均值法（AVERAGE-OF-MAXIMUM）返回最大真实度值出现的变量值的平均值。

## 10.5 语言变量

语言变量既表述事情的重要程度，也表述要做的事情的语境。例如“这个房间很热”，很明确，它代表一个独立于测量系统的主观意见，而且对于它传递的大部分信息听话人都能理解。语言变量在普通的日常活动中都有使用。以速食汤粉的制作为例，其说明便充满着语言参考：把水烧开，不停地搅动，降低温度；盖上一部分，慢煮，然后隔一会儿搅动一下。这些都是在做汤语境中的语言变量，速食汤粉生产厂家相信这些说明虽然简单、模糊，但它能清楚地告诉消费者如何成功地做汤。

下面的例子说明了语言变量（斜体表示）可以在应用软件中正式定义和使用的几种方式。

- 1) 将一杯起酥油和 $2/3$ 杯白糖充分混合。
- 2) 添加一茶匙香草并不停地搅动。
- 3) 打两个鸡蛋，用中等速度搅成泡沫状。
- 4) 慢慢加入两杯面粉。

例如，一位顾客进入商店，想以最优惠的价格买几件小饰品，售货员给出了许多参数制约的价格。这个假想的例子，存在如下制约条件：

- 饰品的成本。
- 正常的涨价幅度。
- 上架时间。
- 存储期。
- 二者之间合作关系持续时间。
- 顾客的支付历史。
- 销售数量。
- 再来消费的可能性。

计算机记录会给这上面的许多参数提供硬性的数据（精确数据）。但仍然有必要给顾客报出一个正常价格的折扣，这一折扣价格可以通过一些组合数据来计算。

这笔交易中，除了“再来消费的可能性”之外，所有参数都是精确数据，都可通过一个组织良好的数据库提供的信息来精确定义。各个参数是怎样影响最终的报价折扣呢？图 10-6a 和图 10-6b 中的表格根据它们哪里重要以及为什么重要的角度对这些参数进行了总结。

所有这些重要条目都依赖语境。可以这么说，“如果数量多，那么利润更多”。在数量语境中数量多是有意义的，更多是利润的结果。在此，给出了关于规则的一个例子。如下的一组全部

规则，是给潜在顾客定义折扣的逻辑。

IF shelf time	IS long	THEN discount IS large
IF shelf time	IS short	THEN discount IS low
IF shelf time	IS short	THEN discount IS high
IF shelf time	IS long	THEN discount IS normal
IF quantity	IS small	THEN discount IS none
IF quantity	IS large	THEN discount IS large
IF quantity	IS huge	THEN discount IS high
IF customer	IS new	THEN discount IS special
IF customer	IS recent	THEN discount IS normal
IF customer	IS long term	THEN discount IS large
IF shelf life	IS short AND Shelf life	IS long THEN discount IS deep

这组规则共11条（或许六七个），可以用来给很多不同产品计算报价。这一规则集合各自描述了销售参数和所提供的折扣间的关系，当这些规则都被估计到的时候，它们会为折扣提供一个权值。

销售参数	重要程度	为什么重要
上架时间	长，短	库存成本
储存期	短，长，永久	达到储存期后，产品价值损失
支付	即时支付，正常，最后结账，拖欠	以往的支付情况，作为与其做生意的一部分成本
数量	少，一般，多，很多	销售大量产品能节省店面开支，增加利润
顾客	新顾客，短期顾客，长期顾客	新顾客需要特殊对待，而长期客户则需要诚信

a) 精确测量语言变量

销售参数	重要程度	为什么重要
回头客	是，可能是，不是	产品和顾客
潜在客户		依赖性

b) 模糊的、主观测量的语言变量

图 10-6

### 10.5.1 使用语言变量

IF room IS cold THEN heat IS on;

IF room IS hot THEN heat IS off;

具有此功能的简单温度调节器已经使用了一百多年。为什么语言变量和模糊逻辑需要进行转换？使用热和冷这些模糊术语如何估计精确的温度值？热和冷到底是什么？

温度控制问题听起来很简单：测量房间的温度，用两个模糊逻辑规则，然后控制加热房间的炉子。当冷和热的意义不是严格相对时，结果会更复杂，而且更有用。

语言变量把语言条件和精确变量联系在一起。精确变量是大部分计算机程序使用的变量，是一个纯粹的数值。另一方面，语言变量具有比例特性，所有软件实现的语言变量都用0~1之间的分数来表示。

前面的例子中，房间和加热器是精确变量，而热、冷、开、关都是语言变量，语言变量开和

关分别代表精确变量加热器的 1 和 0，语言变量热和冷代表了与精确变量房间相关的一个值的变化。图 10-7 展示了这种关系。

大多数语言变量可以在软件中通过四点坐标来表示。精确变量“房间”与语言变量“热”联系在一起，可以通过图中四个孤立的点来定义。

```
LINGUISTIC room TYPE unsigned int MIN 0
MAX 100
{
    MEMBER HOT{ 50,72,100,100 }
}
```

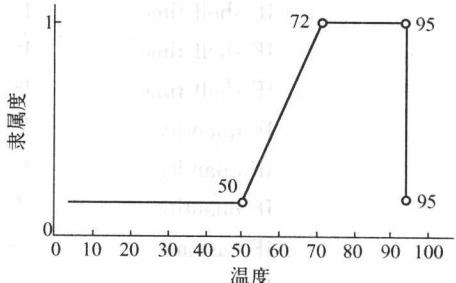


图 10-7 语言变量 HOT

有很多文献是关于语言变量的表示问题的，但是大部分应用都是使用四点表示来实现。人们对于用平滑的曲线表示语言变量的精度有争议，而且因为计算强度问题反对平滑曲线法。四点表示法的最大误差在其拐角处。

### 10.5.2 模糊规则剖析

```
IF room IS cold THEN heat IS on;
```

每个模糊规则都包含两个部分——断言（或谓词）和结论。断言决定了规则的权值或者真实度，房间很冷的结果是位于模糊值 0 和 1 之间的隶属度值。

断言部分的隶属度会加权在模糊规则的结论部分。在自然语言中，打开加热器的紧迫性是由该房间有多冷来决定的。单个模糊规则不能给予精确的比较或动作，然而，多个有竞争性的规则却可以。

### 10.5.3 语言变量的逻辑组合

布尔表达式可以通过逻辑组合生成一个布尔结果，该结果是逻辑表达式的组合运算结果。与之类似，语言变量也是一样。语言变量可以通过或（OR）、与（AND）、非（NOT）运算符来组合。例如用 C 语言编写的预定义语句，可以在应用代码中直接使用，如下：

```
#define F_OR (a,b) ((a) > (b) ? (a) : (b))
#define F_AND (a,b) ((a) < (b) ? (a) : (b))
#define F_NOT (a) (F_ONE + F_ZERO - a)
```

如图 10-8 所示，模糊或取参数中的最大隶属度，模糊与取参数中的最小隶属度，模糊非的取值在参数值与模糊 1 之间。如果语言变量的取值减少到只有 0 和 1 两个数，语言变量控制的逻辑定义就同于传统的布尔逻辑。

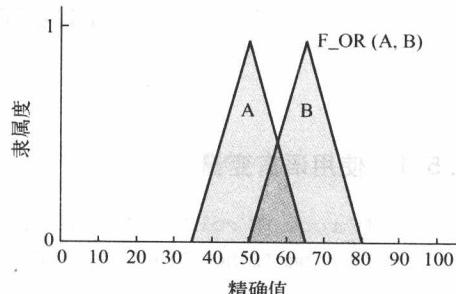


图 10-8 模糊或运算符 (F\_OR)

### 10.6 PID 控制器

经典的比例积分微分（Proportional Integrating Derivative, PID）控制器产生的输出变量是三项之和：第一项是绝对误差与一个常数的乘积，第二项是误差变化率与第二个常数的乘积，第三项是累积误差与另一个常数的乘积。PID 控制系统的一般方程为：

$$mv = (pe \times K1) + \frac{\partial pe}{\partial t} \times K2 + (\sum pe \times K3)$$

PID控制系统三个部分中的每一项都是为了减小误差、预测潜在的误差条件以及克服累积误差。这一描述与用语言变量实现PID控制系统类似。

当利用这种方法实现一个基于语言变量的PID控制器时，可以根据系统误差大小使用三种不同的控制策略。如果误差很大，则主要由误差值单独驱动输出变量。如果误差较小，则由误差变化速率来确定输出变量。对于误差非常小的情况，控制器输出主要由误差的积分来支配。

利用语言变量可以使创建的控制系统的系统常量具有更宽的变化范围。很多显示世界的控制应用都是非线性的。例如飞机控制系统、发动机控制器、食品和化学过程等，在日常使用中，这些系统的系统参数差别很大。

### 10.6.1 时间语言变量

许多应用都会用到语言术语“一天的时间”，但是具体的实现却千差万别。举一个家庭环境应用的例子，把一天分成0.1 h 的时间段，把一天的精确时间存储在单个字节中（编程数据存储），下面的定义创造性地使用了语言变量。

精确小时是一个在午夜复位的循环数字系统。语言变量 day 的定义比较传统：day 始于早晨5:30，而真正的 day 始于早晨6:30；day 持续到下午5:30，而直到下午6:30才真正结束。有了 day 的定义，语言变量 night 的定义就可以不那么复杂，可以通过一个模糊函数 hours IS NOT day 来定义。可见，一个新的语言变量可以通过其他已定义的语言变量来定义。nightsb 语言变量可以通过 night 和 evening 变量进行定义。

```
//0.1 小时的时间间隔把一天分成0……240个时段
LINGUISTIC hours TYPE char MIN 0 MAX 240
{
    MEMBER day {55,65,175,185}
    MEMBER night {FUZZY {hours IS NOT day}}
    MEMBER morning {50,60,190,200}
    MEMBER evening {160,170,190,200}
    MEMBER nightsb {FUZZY {hours IS night AND hours IS NOT evening}}
}
```

### 10.6.2 语言变量比较

实际应用中，许多明确相等的比较被以数据范围说明的模糊比较取代了。模糊比较以三个数值为基础，即单位、失效时的范围（delta）以及当前的变量值。delta 是与当前值的距离，该数值对于停止当前比较来说十分重要。例如这样的定义：在各种情况下 delta 值返回一个模糊 0 或模糊 1，无论偏离中心值多远都不会改变结果。下面对模糊 EQUAL（见图 10-9）给出了一个易于比较实现的定义，如图 10-9 所示。

F\_EQ 的参数主要有 v、cp、delta，v 代表测试中的精确变量，cp 代表中心值，delta 是在模糊比较中离开中心值的距离。F\_EQ 函数可以在任何接受语言变量的表达中使用，比如如下 C 语言代码：

```
DOMtype F_EQ(v,cp,delta)
```

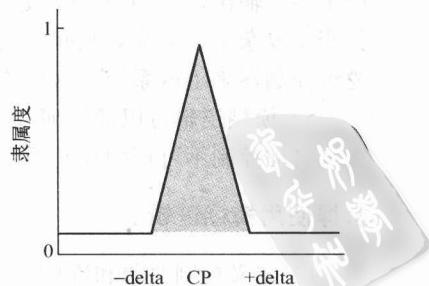


图 10-9 F\_EQ(模糊 EQUAL)

```

        if(m > delta) return (F_zero);
        return ((m/delta) * (F_one - F_zero));
    }
}

```

事实上，cp 和 delta 可以用来声明一个匿名的语言变量。这种方法可以被扩展到所有通常的算法比较中。

语言变量提供了一个规范化的数字系统，其分辨率取决于具体应用的系列要求。语言变量为描述不同策略的比较规则提供了一种自然流畅的转化；语言规则旨在解决问题，而不是分析问题。

在传统的嵌入式微控制器中，语言变量可以较好地实现与使用，而且它们一般不像选择性应用实现那样计算强度大。应用数据规范化是计算强度减小的主要原因，语言变量可以方便地与传统应用软件结合。

## 10.7 模糊逻辑应用

模糊逻辑需要处理一些数字参数，比如某些显著误差和显著的误差变化率，但是这些数据的精确值并不是很重要，除非要求快速响应。在这种情况下，要通过经验来调整数据。例如，一个简单的温度控制系统可以用一个温度反馈传感器，从输入信号中减去反馈数据来计算误差，然后分时产生误差斜率，即误差变化率，以后称之为 error-dot。

这个控制器采用华氏温度 (F)，而且认为  $2^{\circ}\text{F}$  是小误差，而  $5^{\circ}\text{F}$  是大误差。误差变化率 error-dot 以  $^{\circ}\text{F}/\text{min}$  为单位，认为  $5^{\circ}\text{F}/\text{min}$  是小误差变化率，而  $15^{\circ}\text{F}/\text{min}$  是大误差变化率。这些值不一定必须是均匀的，而且系统优化性能时可以进行“调节”。一般地，模糊逻辑要求较宽，很可能使系统在第一次工作时就不需要任何调节。

模糊逻辑独有的特点使它成为许多控制问题的极佳选择，如下。

- 1) 具有本质的鲁棒性，因为它不需要精确、无噪声的输入，如果反馈传感器停止工作或者损坏之后，它可以通过程序控制安全地停止工作。

- 2) 因为模糊逻辑控制器处理采用用户定义规则控制目标控制系统时，可以方便地调节控制器以提高或显著改善系统性能。新传感器可以只通过产生适当的控制规则，就能很容易加入系统。

- 3) 模糊逻辑并不局限于几个反馈输入和一到两个控制输出，也没有必要测量或计算实现系统的变化率参数。系统只需要任意传感器给出系统动作或反应的一些指示数据就足够了。这使得传感器可以降低成本和精确度，从而保证整个系统的复杂度和成本降低。

- 4) 由于是基于规则的运算，系统可以处理任意合理数量 (1~8 个或者更多) 的输入，并且产生多个输出 (1~4 个或者更多)。但是，如果一个系统有太多输入输出，对规则库的定义很快就变得复杂了，因为规则间的关系也需要定义。把控制系统分成几个小块，然后用几个小的模糊逻辑控制器来组成系统，这样会使系统性能更好，各个小系统负责更小的部分。

- 5) 模糊逻辑可以控制那些很难或不可能建立数学模型的非线性系统。这给那些通常看上去通过自动控制不可行的控制系统打开了思路。

### 怎样使用模糊逻辑

- 1) 定义控制对象和准则：要控制什么？要控制此系统必须要做什么？需要什么样的响应？系统不工作的可能状态是什么？

2) 定义输入输出的关系，并且给模糊逻辑系统选择最少的输入变量（典型的误差或者误差变化率）。

3) 使用基于规则的模糊逻辑结构，把控制问题分解成一系列 IF X AND Y THEN Z 的规则。这些规则定义了给定系统输入条件下的期望输出响应，规则的数量和复杂度取决于待处理的输入参数的数量以及与每个参数相联系的模糊变量的数目。如果可能，至少使用一个变量及其时间微分。虽然可以用一个变化率未知的瞬时误差参数，但这会降低系统减小阶跃输入超调的能力。

4) 建立模糊逻辑隶属度函数，定义在规则中使用的输入输出术语的含义或数值。

5) 如果用软件实现，除非把规则写入模糊逻辑硬件系统中，否则要创立必要的预处理和后处理模糊逻辑程序。

6) 进行系统测试、结果评价、规则和隶属度函数调整以及重新设置等，直到得到满意的结果。

把模糊变量看作语言对象或者单词，而不是数字。传感器输入是名词（例如：温度，位置，或者速率）。因为误差只是差异，可以用同样的方式来看待它。模糊变量本身就是修饰变量的形容词。在最小情况下，每个参数可以仅有“正值”、“零值”和“负值”变量。除此之外，还可以使用诸如“非常大”和“非常小”等附加范围，扩展高非线性程度条件的响应程度，但这在系统中不是必须的。

## 10.8 规则矩阵

误差（输入值减去反馈值）和 error-dot（误差变化率）的模糊参数由形容词“负”、“零”和“正”来修饰。为了把这种情况图示出来，一种最简单的实现是一个  $3 \times 3$  的矩阵（见图 10-10）。矩阵的列从左到右代表“负误差”、“零误差”以及“正误差”等误差输入，矩阵的行从上到下代表“负”、“零”、“正”等误差变化率输入。这个平面结构叫做规则矩阵，其输入条件有两个，即“误差”和“误差变化率”，还有一个输出响应（在每行和每列的交叉处）。在这种情况下，有九个可能的逻辑输出响应。

规则矩阵通常有奇数行和奇数列，以实现一个行与列的“零”中心区域，但这并不是必要的。只要中心任何一边的函数有重叠，而且输出的连续抖动是可接受的，就可以不需要“零”中心区域。因为“零”区域与输出“无变化”相关，没有这个区域的响应会导致系统不停地寻找“零”。

列数和行数也可以不一样，这出现在需要大量输入值的时候。最多可能的规则数只是很多行和列的输出，但是没必要定义所有这些规则，因为在实际运算中，有些输入条件可能一直不会出现。该结构的首要目标是在有效控制系统的情况下，制定可能输入的范围。

### 10.8.1 模糊逻辑的实现

实现模糊逻辑的第一步是确定要控制什么和怎样控制。例如，假设要设计一个简易的比例温度控制器，如图 10-11 所示，该控制器有一个电加热部件和一个变速降温风扇，正输出信号会实现  $0 \sim 100\%$  的加热，而负输出信号会实现  $0 \sim 100\%$  的制冷，通过适当平衡和控制加热和制冷这两个装置来实现环境温度控制。

误差		
误差变化率		
负	零	正
N*N	N*Z	N*P
Z*N	Z*Z	Z*P
P*N	P*Z	P*P

图 10-10  $3 \times 3$  规则矩阵

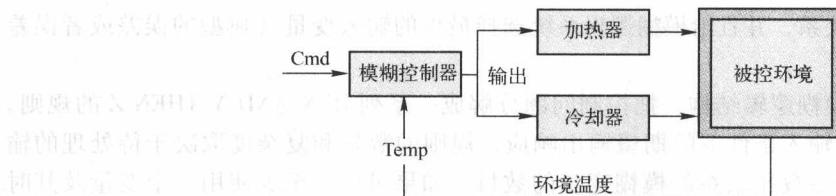


图 10-11 简易模糊逻辑控制系统

变量定义如下：

$Cmd$ : 设定温度

$Temp$ : 控制环境中反馈传感器的测量温度

$Error$ :  $Cmd - Temp$  (正值代表太冷, 负值代表太热)

$Error.\dot{}$ : 误差变化率 (正值代表正在变热, 负值代表正在变冷)

$Output$ : 加热、无变化或制冷

有必要建立一个具有一定含义的系统来表示矩阵中的语言变量：

"N" 表示 "负" 误差或 "负" 误差变化率输入值

"Z" 表示 "零" 误差或 "零" 误差变化率输入值

"P" 表示 "正" 误差或 "零" 误差变化率输入值

"H" 表示 "加热" 输出响应

" - " 表示当前输出 "无变化"

"C" 表示 "制冷" 输出响应

使用这些术语, 可以定义最小数量的输入组合以及相应的输出响应。对于一个有加热和制冷输出响应的  $3 \times 3$  矩阵, 九个规则都需要定义, 与每个规则的输出响应相关的语言变量规则结论都被转移到矩阵中。由图 10-12 可以看出, 在典型的控制系统中, 系统趋于稳定时设定值与误差之间的关系。本例中的定义见图 10-18。

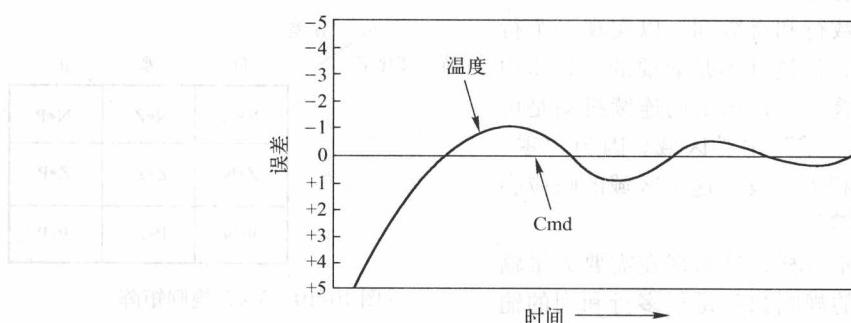


图 10-12 典型的控制系统响应

该控制系统的规则定义举例如下：

INPUT #1: ("Error", positive(P), zero(Z), negative(N))

INPUT #2: ("Error.\dot{}, positive(P), zero(Z), negative(N))

CONCLUSION: ("Output", Heat(H), No Change(-), Cool(C))

INPUT#1: System Status

Error = Command-Feedback

P = Too Cold, Z = Just right, N = Too hot

INPUT#2 System Status

Error-dot =  $d(\text{Error}/dt)$

P = Getting hotter, Z = Not changing, N = Getting colder

OUTPUT Conclusion and System Response

Output H = Call for heating, - = Don't change anything, C = call for cooling

语言规则描述的控装置系统有两部分组成，即前提模块（在 IF 和 THEN 之间）和结论模块（在 THEN 后面）。根据具体情况决定是否有必要估算每个可能的输入组合 ( $5 \times 5$  矩阵)，因为有些输入几乎不会发生。通过做这种估算，可以减少规则数量，从而简化处理逻辑，而且很可能会提高模糊逻辑系统的性能。这种估算一般由经验丰富的操作员来完成。

把九条规则的结论转移到矩阵之后，矩阵显然是对称的。这表明该系统是符合实际、性能良好的（线性）系统，但也不能保证必定这样。这种实现在有些控制问题中可能显得过于简单，但是它说明了这个过程。如果想等到期望的系统响应，可能还会添加误差和误差变化率。这会增加规则库的规模和复杂度，但也会提高系统控制质量，如图 10-13 所示。

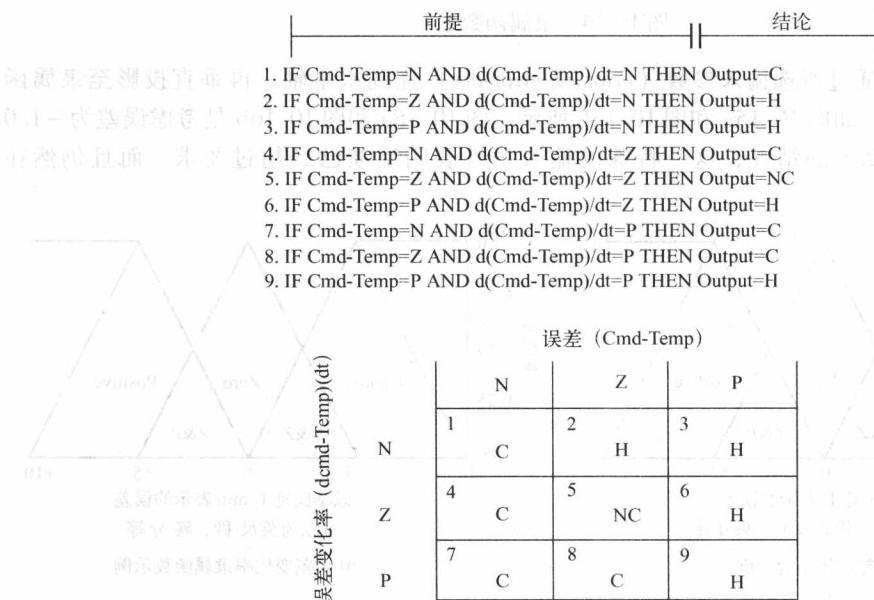


图 10-13 规则结构和规则矩阵

### 10.8.2 隶属函数

隶属函数是各个输入幅度的图形表示，它给每个待处理的输入加一个权值，定义了各输入之间的函数重复程度，而且最终决定输出响应。模糊规则将输入隶属度当作权重因子，决定其最后输出结果模糊集上的影响。隶属函数一旦确定并与赋予数值结合，它们就通过去模糊化为驱动系统提供精确输出。联系各个输入和输出响应的隶属函数是不同的，需注意以下几点：

- 形状。**三角形比较常用，还有用钟形、梯形、半正矢形、指数形等。更复杂的函数也有可能实现，但是需要更大量的计算。
- 肩部。**如果是一个外部函数，高度固定在最大值处。肩形函数在通过中心点时取 1.0。

中心点是隶属函数图形的中心；重叠部分即 N&Z, Z&P，一般是宽度的 50%，还可以再少。

图 10-14 说明了三角形隶属函数的特征，由于其在数学方面比较简单，故以此举例。其他形状也可以用。

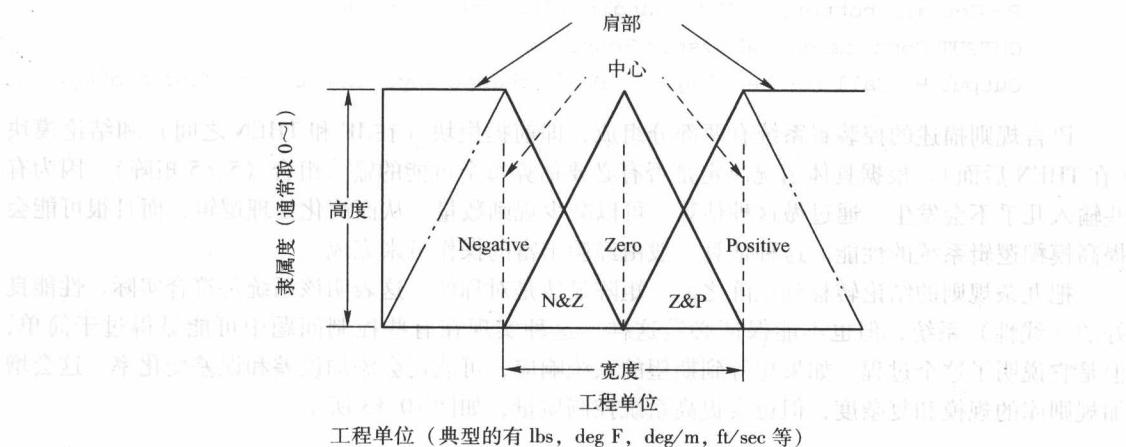
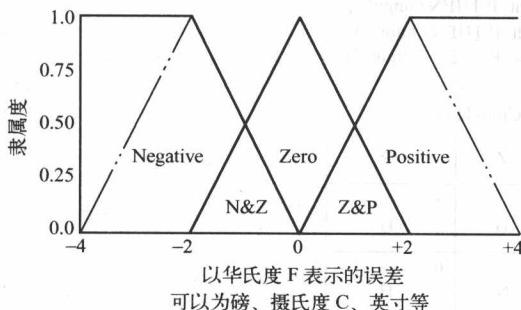
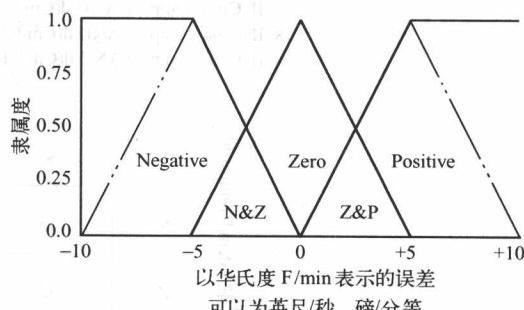


图 10-14 隶属函数

隶属度 (DOM) 通过所选输入参数 (error 或 error-dot) 连到水平轴，再垂直投影至隶属函数的上边界处来得到。如图 10-15a 和图 10-15b 所示。图 10-16a 和图 10-16b 是考虑误差为 -1.0 以及误差变化率为 +2.5 的情况，这一特殊的输入条件表明反馈已经超过要求，而且仍然在增加。

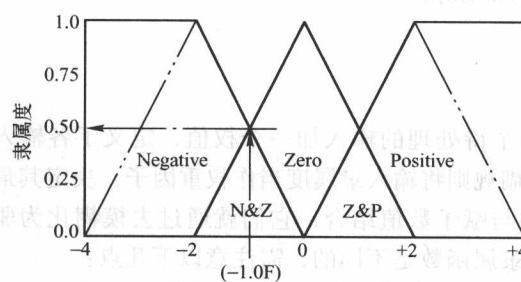


a) 误差隶属函数示例



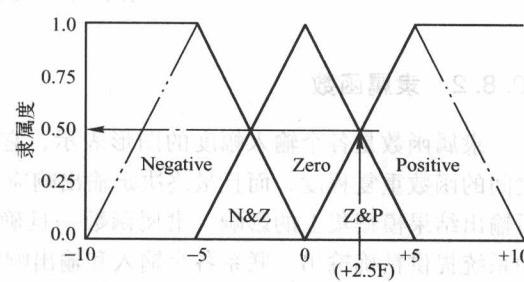
b) 误差变化率隶属函数示例

图 10-15



以华氏度 F 表示的温度误差

a) 误差隶属函数



以 F/min 表示的温度误差变化率

b) 误差变化率隶属函数

图 10-16

误差为 -1.0 的隶属度向上投影到 negative 函数和 zero 函数重叠部分的中间，因此结果为 negative 隶属度 = 0.5、zero 隶属度 = 0.5。只有与 negative 和 zero 误差联系在一起的规则才真正适用于输出响应。这样便只要选择规则矩阵左边和中间的列。

对于误差变化率为 +0.25 的情况，可以看出 positive 隶属度 = 0.5，zero 隶属度 = 0.5。这样便选择了规则矩阵的中间行和最下面一行。可以看到，只有规则矩阵中两个区域重叠的部分，即只有矩阵左下角的  $2 \times 2$  方阵才会产生非零的输出结果。因为规则中逻辑与的存在，使得其他规则权值为零。

有一个独特的隶属函数与每个输入参数都有关系。该隶属函数与每一输入数值的权重因子和有效值有关。这些权重因子决定了影响的程度或者每个有效规则的隶属度。通过计算每个有效规则的隶属权值的逻辑输出，就可以得到一组模糊输出响应幅度，然后就是将这些输出响应进行合并和去模糊化。每个有效规则的模糊输出响应幅度必须通过某种方式进行组合和处理产生一个精确（去模糊化的）输出。

当系统有输入时，规则库会根据输入进行计算。前提模块 (IF X AND Y) 测试输入并产生结论。有一些规则的结论模块 (THEN Z) 条件满足了，而其他的则没有被满足。这些结论被组合在一起形成一个逻辑总和。随后，这些结论进入推理过程，在推理过程中要确定每个输出的隶属函数的触发强度（在 0~1 之间）。

### 10.8.3 隶属度输入

回到规则上，插入图 10-17 所示的隶属函数权值，“误差”选择规则 1, 2, 4, 5, 7 和 8，而“误差变化率”选择规则 4~9，所有规则的“误差”和“误差变化率”被组合成一个逻辑输出 (LP 或 AND 中的较小者)。

```
"error" = -1.0: "negative" = 0.5 and "zero" = 0.5
"error-dot" = +2.5: "zero" = 0.5 and "positive" = 0.5
ANTECEDENT & CONSEQUENT BLOCKS (e=error,
er=error-dot or error-rate)
```

图 10-17 隶属函数权值

所选的这 9 个规则中，只有 4 个（规则 4, 5, 7, 8）触发，或者说有非零结果。这使得模糊输出响应只有“制冷”(cooling) 和“不变”(No Change)，这些结果必须经过推理、组合和去模糊化，以返回真正的精确输出。在下列规则列表中，e 代表误差 error，er 代表误差变化率 error-dot。

1. If (e < 0) AND (er < 0) then Cool 0.5 & 0.0 = 0.0
2. If (e = 0) AND (er < 0) then Heat 0.5 & 0.0 = 0.0
3. If (e > 0) AND (er < 0) then Heat 0.0 & 0.0 = 0.0
4. If (e < 0) AND (er = 0) then Cool 0.5 & 0.5 = 0.5
5. If (e = 0) AND (er = 0) then No\_Change 0.5 & 0.5 = 0.5
6. If (e > 0) AND (er = 0) then Heat 0.0 & 0.5 = 0.0
7. If (e < 0) AND (er > 0) then Cool 0.5 & 0.5 = 0.5
8. If (e = 0) AND (er > 0) then Cool 0.5 & 0.5 = 0.5
9. If (e > 0) AND (er > 0) then Heat 0.0 & 0.5 = 0.0

使用 AND 运算符对输入进行逻辑组合，对所有期望输入产生相应输出响应。接下来，有效

结果都合并到各隶属函数的逻辑和中。这样，就可以计算出每个输出隶属函数的触发强度。最后，在去模糊化过程中把这些逻辑和组合起来产生精确输出。

#### 10.8.4 推理

上述例子中，最后一步完成的是确定每个规则的触发强度。在最终结果中，规则 4, 5, 7 和 8 的触发强度都是 50%，而规则 1, 2, 3, 6 和 9 没有触发，即触发强度是 0。每个规则的逻辑输出必须在通过产生精确输出的去模糊化过程前进行组合或推理（利用 MAX-MIN 方法、MAX-DOT 方法、AVERAGED 方法、RSS 方法等）。

MAX-MIN 方法测试每个规则的值，然后选出输出值最高的，把隶属函数下区域的模糊重心的横坐标作为输出。这种方法没有组合所有可用规则的结果，但却产生了一个连续的输出函数，而且容易实现。

MAX-DOT 或者 MAX-PRODUCT 方法会改变每个隶属函数的大小来适应其每个峰值，并且把隶属函数下区域的模糊重心的横坐标作为输出，必须通过缩小隶属函数来使得其峰值等于各自的函数（negative, zero 和 positive）的幅值。这种方法组合了所有有效规则的影响，并且会产生一个连续平滑的输出。

AVERAGING 方法虽然不能给更多规则提供递增的权值来支持每个隶属函数的输出，但它却是一种很有效的方法。例如，如果触发了 3 个 negative 规则，而只触发了 1 个 zero 规则，均值不会反映出差别，这是因为均值都是 0.5。每个函数都影响权值，可以计算重叠区域的模糊重心。

RSS (root-sum-square) 方法结合了所有可用的规则，按函数的幅值来改变函数的大小，而且计算重叠区域的模糊重心。这种方法在数学上比其他方法都复杂，这里利用该方法举例，因为它能给所有的有效规则赋予最合理的权值。

### 10.9 去模糊化

RSS 方法包括了所有有用的规则，因为只有很少的隶属函数和输入输出联系在一起。对于正在讨论的例子，误差 -1.0 和误差变化率 +2.5 选择了 negative 和 zero 的输出隶属函数区域。R1 ~ R9 这些可能的规则中，各自的输出隶属函数强度（范围：0 ~ 1）如下：

$$\text{"negative"} = (R1^2 + R4^2 + R7^2 + R8^2) \cdot (\text{Cooling}) = (0.00^2 + 0.50^2 + 0.50^2 + 0.50^2)^{.5}$$

$$= 0.866$$

$$\text{"zero"} = (R5^2)^{.5} = (0.50^2)^{.5} \cdot (\text{no change}) = 0.500$$

$$\text{"positive"} = (R2^2 + R3^2 + R6^2 + R9^2) \cdot (\text{Heating}) = (0.00^2 + 0.00^2 + 0.00^2 + 0.00^2)^{.5}$$

$$= 0.000$$

组合推理工程的输出结果以及计算区域的模糊重心，可以通过去模糊化来实现数据的精确输出。各个输出隶属函数的权值乘以各自输出隶属函数的中心值，然后求和，再以求出的和除以加权隶属函数强度的和，所得出的结果就是精确输出值。需要注意的一点是，由于 zero 中心在零点，任何零强度自动为 0。如果 zero 函数的中心偏离零点（在实际系统中就是加热效果和制冷效果不完全相当）：

$$\frac{(\text{neg_center} * \text{neg_strength} + \text{zero_center} * \text{zero_strength} + \text{pos_center} * \text{pos_strength})}{(\text{neg_strength} + \text{zero_strength} + \text{pos_strength})} = \text{OUTPUT}$$

这个因素会对输出有影响，通过适当替换可以产生输出：

$$\frac{(-100 * 0.866 + 0 * 0.500 + 100 * 0.000)}{(0.866 + 0.500 + 0.000)} = 63.4\%$$

在图 10-18 中, 所标出的区域重心的横坐标可以看作规一化的精确输出。-64.3% 这一数值 (64.3% 制冷) 看上去符合逻辑, 主要是因为特殊的输入条件 ( $\text{error} = -1$ ,  $\text{error-dot} = +2.5$ ) 表明反馈超过了需要而且仍在增加, 因此制冷是期望的并且也是需要的系统响应。

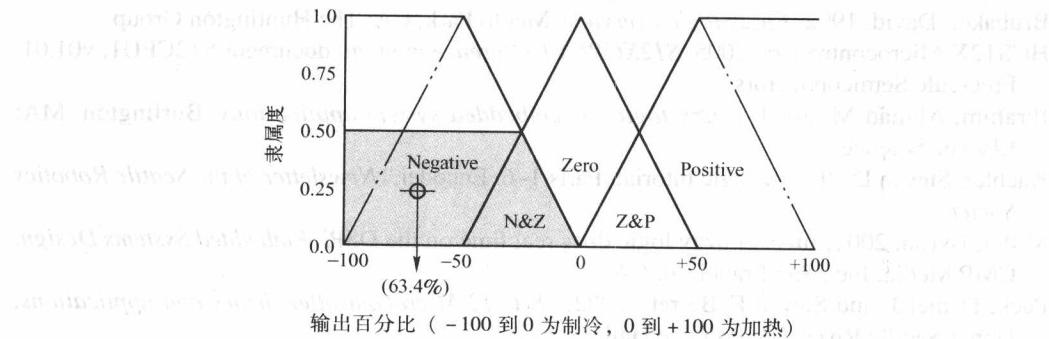


图 10-18 输出隶属函数

## 10.10 调整与提升系统性能

系统调整可以通过几种方式来实现, 比如: 通过改变规则前提或结论, 改变输入和(或)输出的隶属函数中心, 或者给输入和(或)输出函数增加额外的修饰程度, 比对误差、误差变化率, 以及输出响应增加诸如“低”、“中等”和“高”的级别。这些新的级别会产生额外的规则和隶属函数, 会和相近的隶属函数产生重叠, 形成更长的函数和响应范围。系统地讲, 这种方法就是一种主体自我调节。

每个规则的逻辑结果会通过推理给每个输出隶属函数产生一个组合幅值。这可以通过 MAX-MIN, MAX-DOT, AVERAGING, RSS 等方法来完成。一旦推理完成, 这些幅值将被映射到各自的输出隶属函数, 圈定其中的一部分或全部。计算隶属函数重叠区域的“模糊重心”, 并把最终结果作为精确输出, 调节系统规则和系统函数定义的参数来得到可接受的系统响应。

## 习题

1. 为什么模糊逻辑要防止超调?
2. 描述模糊逻辑的 3 个基本组成部分。
3. 举出 5 个非模糊数据的例子。
4. 举出 3 个模糊集的例子。
5. 解释模糊逻辑如何利用常识。
6. 隶属度的含义是什么?
7. 举出 3 个语言变量的例子。
8. 写出 5 个模糊逻辑规则。
9. 什么是去模糊化?
10. 选择一个语言变量, 并用示意图表示。
11. 描述 PID 控制器的主要功能。
12. 举出 2 个显著误差和误差变化率的例子。
13. 在规则矩阵中, 行和列的中心零点取值是多少?
14. 用示意图表示控制比萨炉的模糊逻辑系统。
15. 举一个去模糊化的例子。

## 参考文献

- Banks, Walter. *Linguistic variables: Clear thinking with fuzzy logic*. Byte Craft Ltd., Waterloo, Ontario, Canada.
- Brubaker, David. 1992. *Fuzzy logic overview*. Menlo Park, CA: The Huntington Group.
- HCS12X Microcontrollers. 2005. *S12XCPUV1 reference manual*, document S12CPU1, v01.01. Freescale Semiconductors.
- Ibrahim, Ahmad M. 2004. *Fuzzy logic for embedded system applications*. Burlington, MA: Elsevier Science.
- Kaehler, Steven D. "Fuzzy logic tutorial, Parts 1–6. Encoder," *Newsletter of the Seattle Robotics Society*.
- Miller, Byron. 2004, June. "Fuzzy logic does real time on the DSP." *Embedded Systems Design*. CMP Media, Inc., San Francisco, CA.
- Pack, Daniel J. and Steven F. Barrett. 2002. *68HC12 Microcontroller theory and applications*. Upper Saddle River, NJ: Prentice Hall.
- Petriu, Emil M. *Fuzzy systems for control applications*. School of Information Technology and Engineering, University of Ottawa, Canada.
- Sowell, Thomas. *Fuzzy logic for just plain folks*. Retrieved from <http://www.fuzzy-logic.com/>.
- Wikipedia. *Fuzzy logic*. Retrieved from [http://en.wikipedia.org/wiki/Fuzzy\\_logic](http://en.wikipedia.org/wiki/Fuzzy_logic).

## 8位微控制器

- 本章目标：简单介绍两种流行的8位通用微控制器
- 学习内容：

- MicroChip 的8位微控制器 PIC18F4520。
- ZiLOG 的eZ8 Encore XP F0830 微控制器。

### 11.0 通用微控制器

由于采用了经过验证的体系结构，通用微控制器（General-Purpose Microcontroller, GPM）经受住了时间的考验。通用微控制器的关键特性之一就是它们成熟的体系结构，这样就使其具有确定的产品开发工具市场，进而可以降低设计成本。

通用微控制器的一个缺点是它们的体系结构固定，因此对于特定的设计应用就不能有针对性地设置最小的功能集。因此，生产商设计了众多外围设备功能，能够广泛地适用于各种应用。这样，通过增加产量，并扩大应用范围，便可使得生产成本降低。图 11-1 所示为 PIC 微控制器的应用市场分割图。

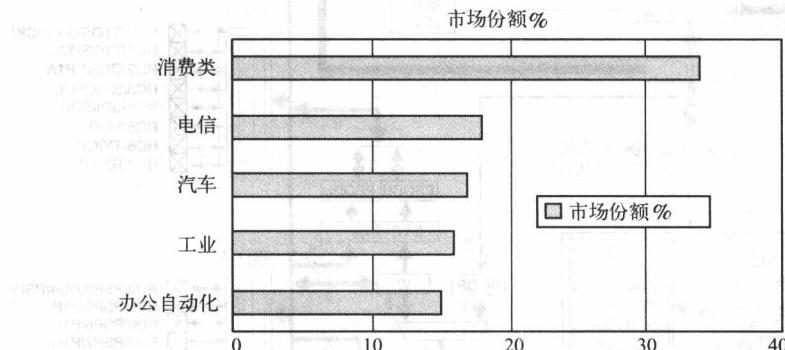


图 11-1 PIC 微控制器的应用市场分割（经微芯科技股份有限公司许可使用）

当大量使用通用微控制器时，产品成本既依赖于设计成本，也依赖于每个芯片的价格。尽管在整个产品生产期间，使用专用芯片可能会降低每个芯片的价格，而使用通用微控制器却可以大大降低前端成本，从而降低商业风险，如图 11-2 所示。当在产品中选择使用定制、半定制或者通用微控制器时，都要综合考虑这些因素。

在使用 GPM 时，还有其他一些需要考虑的因素，包括上市时间和性能。与使用 GPM 相比，使用系统芯片的一个明显优势在于其性能。SoC 可以针对特定的应用进行设计，因此在硬件上具有最优的性能。通常对于关键的功能，GPM 必须依赖软件来实现，这样就会降低其性能。

我们已经看到，除了产量低和设计成本低之外，还有很多因素影响着 GPM 的使用。本章将

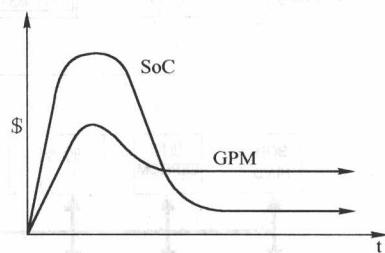


图 11-2 SoC 与 GPM 的成本比较

介绍的两种体系结构具有很好的市场成功经验，并且应用范围比较广，应用量也比较大。

## 11.1 微芯公司 PIC18F4520

该芯片是 PIC18F452 系列产品成功的改进版本，具有 8 位的 Harvard 体系结构，采用了成熟的设计技术，应用范围很广泛。成功的 FLASH 存储器技术使该芯片的使用量较大，且成本很低，使其可以广泛地应用于实际产品中。

图 11-3 给出了 PIC18F4520 的功能框图。除了标准的 I/O 端口之外，芯片还包括一些额外功

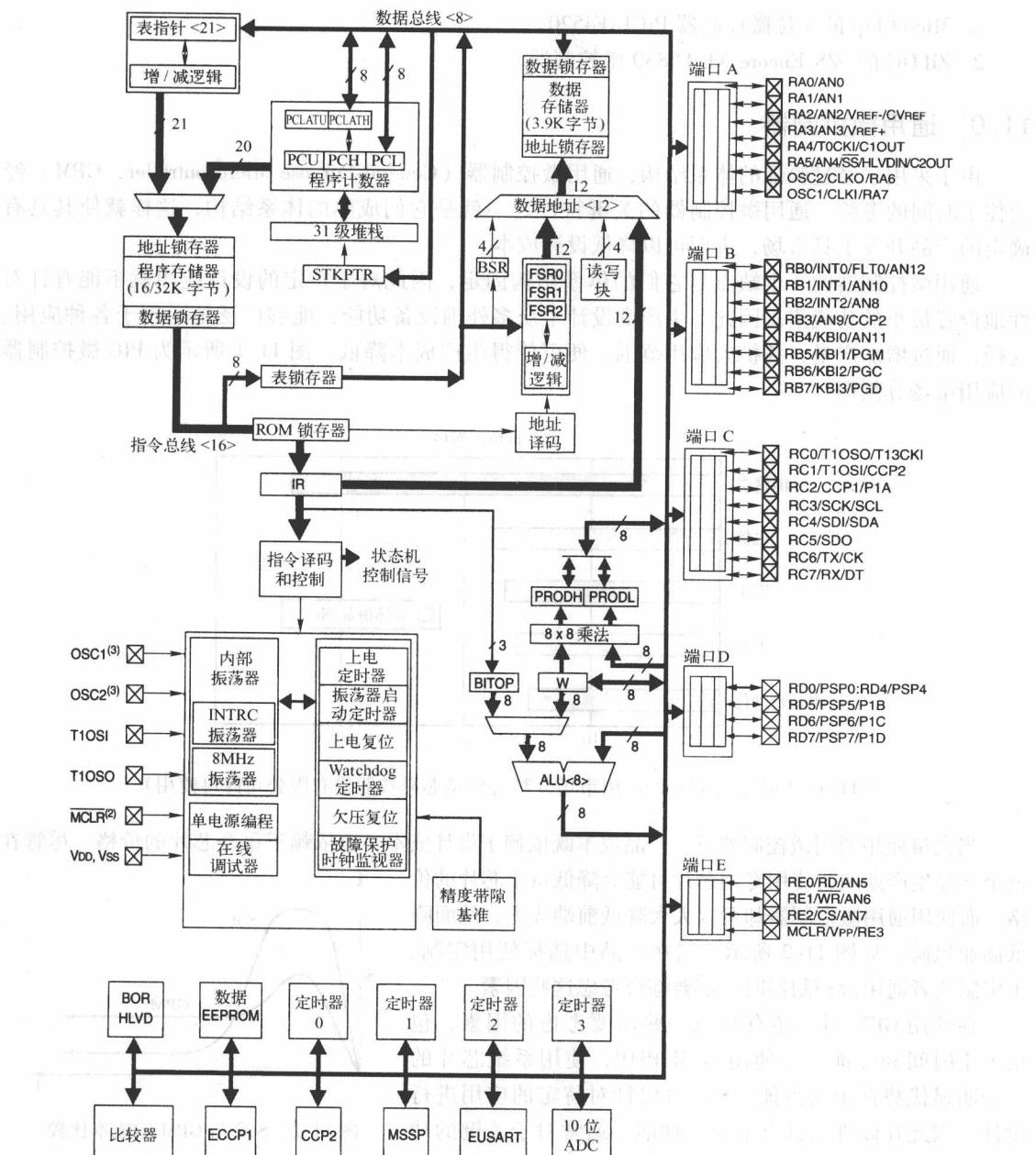


图 11-3 PIC18F4520 的功能框图（经微芯科技股份有限公司许可使用）

能，如多个定时/计数器、A/D 转换器、电可擦除可编程只读存储器（Electrically Erasable Programmable Read-Only Memory, EEPROM）、通用异步收发器（UART）以及捕捉和比较功能等。这些特殊的硬件通过端口引脚的多路复用，在外部实现功能接口。

### 11.1.1 PIC18F4520 Harvard 体系结构

PIC18F4520 系列产品实现了一种经典的 Harvard 体系结构。芯片中有 1 536 字节的数据存储器和 32 758 字节的 FLASH 程序存储器。图 11-4 给出了分离的存储器和它们各自的总线。Harvard 体系结构具有分离的数据和程序总线，可以在每一个读指令周期中读取一个字节的数据。

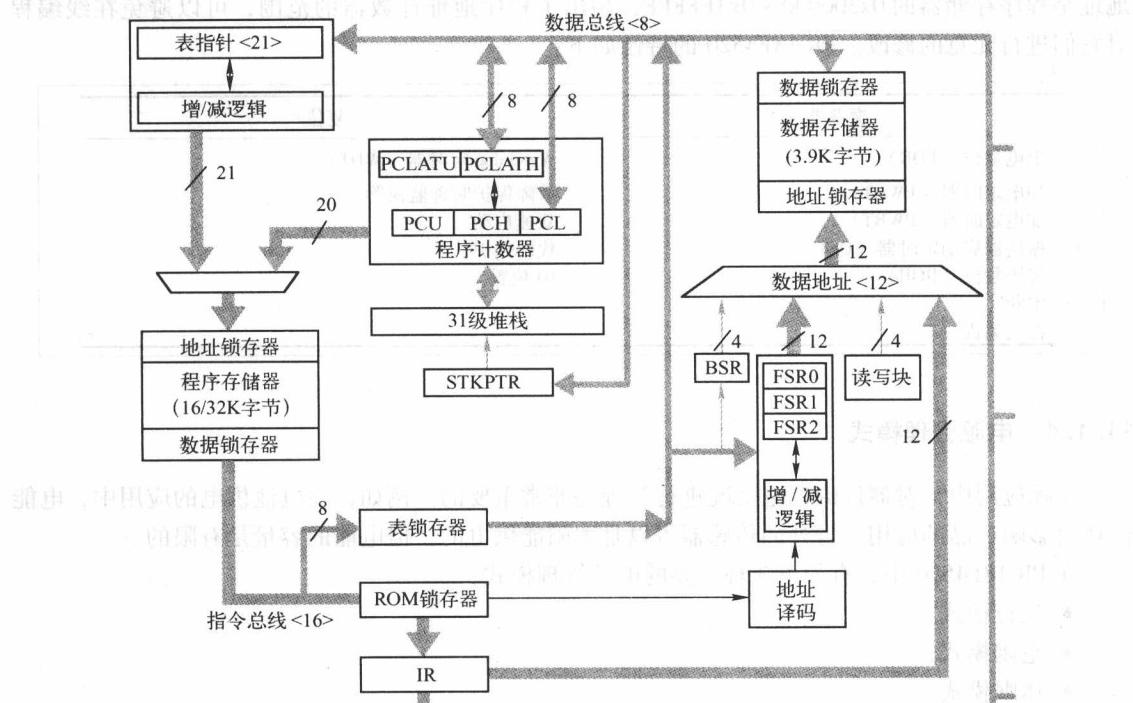


图 11-4 PIC18F4520 Harvard 体系结构（经微芯科技股份有限公司许可使用）

### 11.1.2 指令流水线

PIC18F4520 中设计了一种经典的指令流水线，流水线中的取指和指令译码周期相互重叠。这种两站的流水线实现起来很简单（芯片面积小），然而却能够使指令吞吐量增加一倍。由于只有两站，与 5 站或 7 站的流水线相比，控制逻辑比较简单，开销也小。

图 11-5 给出了流水线实现的原理，展示了取指和执行周期的相互重叠。与所有其他不具备

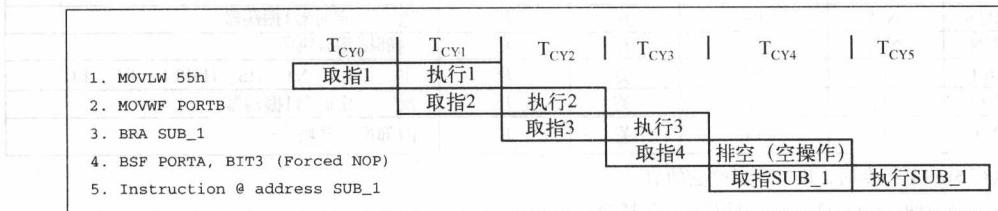


图 11-5 PIC18F4520 两站流水线（经微芯科技股份有限公司许可使用）

前瞻逻辑的流水线一样，当遇到程序流变化时（任何类型的跳转指令），流水线将清空，这时需要插入额外的周期。例如在图 11-5 中，BRA（跳转）指令需要两个指令周期。

### 11.1.3 特性

PIC18F4520 中包含了一些功能模块，可以减少对外部电路的需求。这样可以简化设计，同时又能增强性能。尤其是在设计产品 PCB 时，这个特性非常重要，可以少使用一些外部部件。实际上，此时芯片的 PCB 就是产品的 PCB。

控制器中有一个包含 11 个配置寄存器的寄存器组，可以配置 9 种主要特性。这些寄存器的地址是程序存储器的 0x300000 ~ 0x3FFFFF，超出了程序地址计数器的范围，可以避免在线编程对它们进行无意的修改。PIC18F4520 的特性如下：

振荡器选择	复位
上电复位 (POR)	Watchdog 定时器 (WDT)
加电定时器 (PWRT)	故障保护时钟监视器
加电定时器 (PWRT)	双速启动
振荡器启动定时器 (OST)	代码保护
欠压复位 (BOR)	ID 位置
中断	
在线编程	

### 11.1.4 电源管理模式

实际应用中，对器件的耗电情况进行管理是非常重要的。例如，在电池供电的应用中，电能消耗会影响产品的应用。野外的传感器可以是太阳能供电的，但电能的容量是有限的。

在 PIC18F4520 中，有如下 3 种主要的电源管理模式：

- 运行模式；
- 空闲模式；
- 休眠模式。

这些模式类型定义了给设备的哪些组成部分提供时钟，并且根据所处的模式不同，时钟的速度也不同。在运行模式和空闲模式中，可以使用 3 个输入时钟源的任何一个，而休眠模式则不使用任何时钟源。图 11-6 给出了不同的模式以及个别模式的时钟源和速度。

模式	OSCCON位		模块时钟		有效时钟和振荡源
	IDLEN <sup>①</sup> <7>	SCS1:SCS0 <1:0>	CPU	外设	
休眠	0	N/A	关	关	无——所有的时钟都被禁止
PRI_RUN	N/A	00	开	开	主——LP、XT、HS、HSPLL、RC、EC 和内部振荡器块 <sup>②</sup> ，这是正常的全功耗执行模式
SEC_RUN	N/A	01	开	开	次——定时器1振荡器
RC_RUN	N/A	1x	开	开	内部振荡器块 <sup>②</sup>
PRI_IDLE	1	00	关	开	主——LP、XT、HS、HSPLL、RC、EC
SEC_IDLE	1	10	关	开	次——定时器1振荡器
RC_IDLE	1	1x	关	开	内部振荡器块 <sup>②</sup>

① 执行 SLEEP 指令时，IDLEN 反映它的值。

② 包括 INTOSC 和 INTOSC 后分频器，还包括 INTRC 源。

图 11-6 PIC18F4520 电源管理模式（经微芯科技股份有限公司许可使用）

在运行模式下，内核和外设的时钟保持有效。不同模式之间的区别在于时钟源。在 PRI\_RUN 模式中，时钟源主要来自振荡器输入；而在 SEC\_RUN 模式中，时钟源来自定时器 1 外部振荡器。这样就提供了一种低功耗模式，而同时仍然可以使用高精度的时钟源。

在 RC\_RUN 模式下，CPU 和外设的时钟来自内部 RC 振荡器。这个模式在程序运行时消耗的电能最小。而付出的代价就是时钟精度较低，这对于许多用电池供电的应用来说是比较好的选择。

在休眠模式下，设备并不执行程序指令，时钟源也被切断。这样可以尽可能地节约电能，因为此时并没有执行指令。当发生中断、RESET 或者 WDT 定时时间到等事件时，就会产生唤醒事件。

在空闲模式下，CPU 被停止，而外设仍然处于有效状态。如果长时间（相对来说）没有操作时，就会产生一个中断，使设备进入这个状态，以节省功耗。此时 CPU 并不执行指令，只是等待中断、WDT 定时时间到或者 RESET，来恢复指令的执行。

### 11.1.5 振荡器配置

PIC18F4520 共有 10 种不同的振荡器模式，并可以通过 0x01 配置寄存器的 FOSC3：FOSC0 位进行配置。这些模式如下：

1	LP	低功耗晶体
2	XT	晶体/振荡器
3	HS	高速晶体/振荡器
4	HSPLL	高速晶体/振荡器，PLL 启用
5	RC	外部电阻/电容，Fosc/4 输出到 RA6
6	RCIO	外部电阻/电容，I/O 输出到 RA6
7	INTIO1	内部振荡器，Fosc/4 输出到 RA6，I/O 输出到 RA7
8	INTIO2	内部振荡器，I/O 输出到 RA6 和 RA7
9	EC	外部时钟，Fosc/4 输出
10	ECIO	外部时钟，I/O 输出到 RA6

选择振荡器配置时，应基于成本、时钟精度和应用综合考虑。对于要求成本最小而时钟精度不是很重要的应用，可以使用 RC 和 RCIO 模式。

此外，使用内部振荡器可以减小部件的数量。其中包含了一个振荡器“调整”寄存器，可以通过用户应用程序代码来调整时序。出厂设置为 8 MHz，然而，随着（芯片） $V_{dd}$  和温度的变化，这个数值会发生变化。

### 11.1.6 复位

PIC18F4520 中有以下几个振荡器配置，以适应不同的时钟精度和成本要求。

- 上电复位（Power-On-Reset，POR）。
- 在正常操作中的 MCLR/Reset。
- 电源管理模式中的 MCLR/Reset。
- 看门狗定时器（Watchdog Timer，WDT）复位（在执行过程中）。
- 可编程掉电复位（Brown-out-Reset，BOR）。
- RESET 指令。

- 堆栈满复位。堆栈满时，生成溢出信号，禁止向堆栈写入数据，同时将堆栈指针置零。
  - 堆栈下溢复位。堆栈下溢时，生成溢出信号，禁止向堆栈读取数据，同时将堆栈指针置零。

图 11-7 给出了 RESET 逻辑功能框图的简化图。大多数寄存器不受 RESET 的影响。如果处理器处于休眠模式，则大多数寄存器都不会受到由 WDT 或中断引发的唤醒的影响。如果寄存器中的位直接与唤醒条件相关，则属于例外的情况。

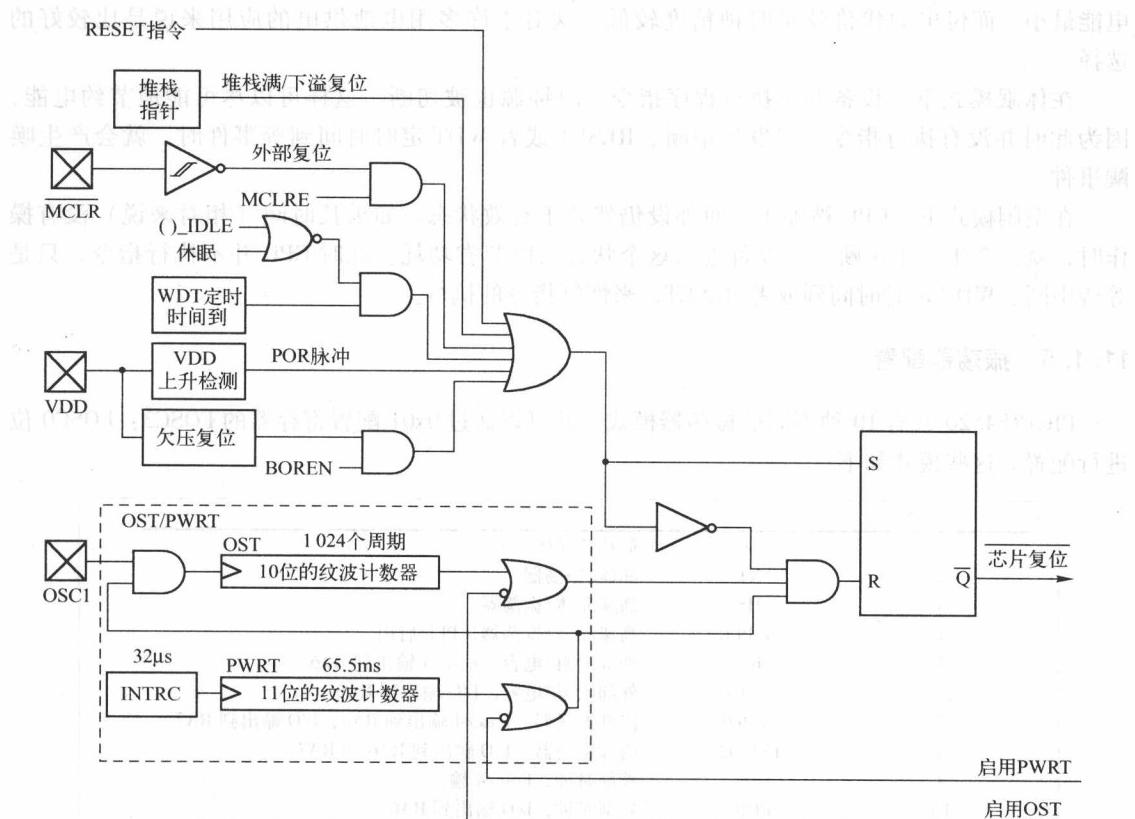


图 11-7 PIC18F4520 片上复位电路模块图（经微芯科技股份有限公司许可使用）

MCLR、WDT 复位、RESET 指令和堆栈复位基本上不会改变控制寄存器，它们将程序流中的寄存器复位到初始状态。POR 和 BOR 会将所有的寄存器完全复位到它们的初始状态。

### 11.1.7 存储器组织

PIC18F4520 有 3 个存储器空间，即程序存储器、数据存储器和外设 EEPROM。如果采用 Harvard 风格，在一个指令周期中，指令和数据存储器都可以进行访问。EEPROM 存储器可作为外设功能处理，处于数据或指令存储器空间之外。图 11-8 给出了基本的程序存储器图。

程序存储器采用 FLASH 技术实现，既可以内部编程，也可以外部编程。这样就允许将 PIC18F4520 设计在 PCB 上出厂，并仍然可以远程进行编程。PIC18F4520 使用 32 K 字节（16 K 字）的 FLASH 实现，超出了 2 MB 的寻址空间。

数据存储器组织为 256 字节的寄存器组，如图 11-9 所示。尽管页选择寄存器（Bank Select Register，BSR）可以寻址 16 页，而实际上只使用 6 页。程序员必须跟踪所选择的页，确保正确地对数据进行读写操作。

定义一个直接存取页后，数据存储器的前 128 个字节和寄存器组 15 的高 128 个字节可以映射到一个页中。直接存取页不使用 BSR，通过设置  $a=0$  对指令进行定义，“迫使”使用直接存取页，而将 BSR 旁路。

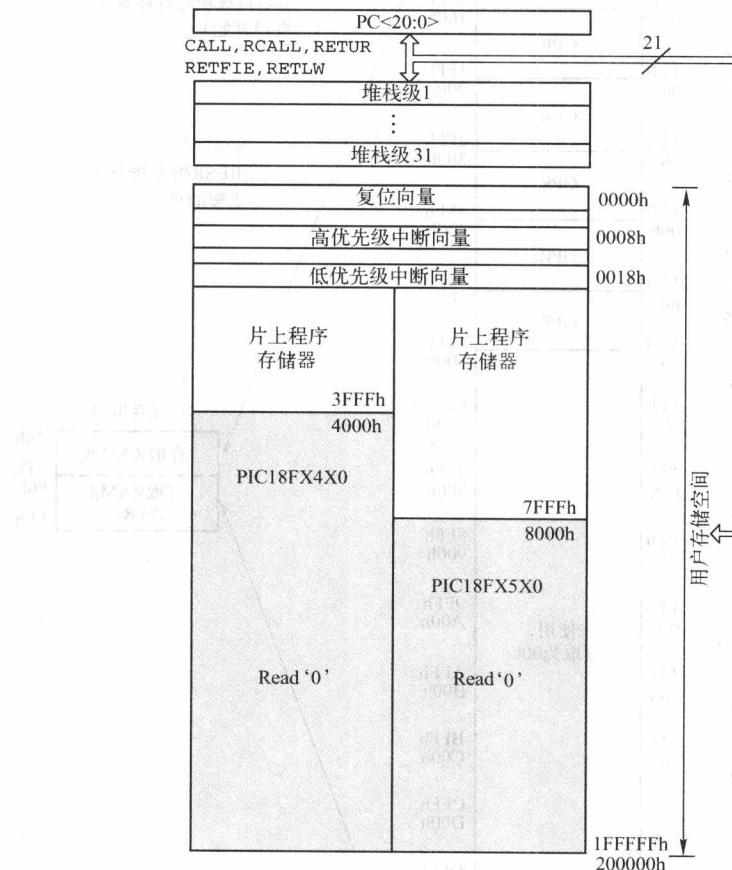


图 11-8 PIC18F4520 程序存储器图（经微芯科技股份有限公司许可使用）

EEPROM 存储器单独寻址，处于程序/数据存储器地址空间之外。EEPROM 通过特殊功能寄存器 (Special Function Register, SFR) 进行访问，应用寻址位于 0x00 ~ 0xFF 之间的 256 个地址。根据操作不同，EEDATA 寄存器保存读或写字节。EECON1 和 EECON2 是进行读和写操作的控制寄存器。

数据记录器可以作为一个应用例子，当数据输入系统中时，存储在 EEPROM 存储器中；一段时间之后，电源可能经过了周而复始的切断和连接，而重要的数据仍然被保存在存储器中。

### 11.1.8 中断结构

PIC18F4520 为外设功能和外部中断功能提供了非常复杂的中断结构，在系统中定义了两种层次的中断优先级，如图 11-10 所示。所有的中断都可以是两种优先级中的任意一种，低优先级的中断向量地址位于 0x0018，高优先级的中断向量位于 0x0008。INT0 ( $RB <0>$ ) 只能被定义为高优先级中断，以便支持向后兼容性。

除 INT0 之外，每个中断都有一个相关的使能位、标志位和优先位。当发生中断后，相应的标志位就被设置为有效，若相应的使能位也被设置为有效，则相应的中断地址向量会产生中断。通过测试中断标志位可以对中断进行查询。

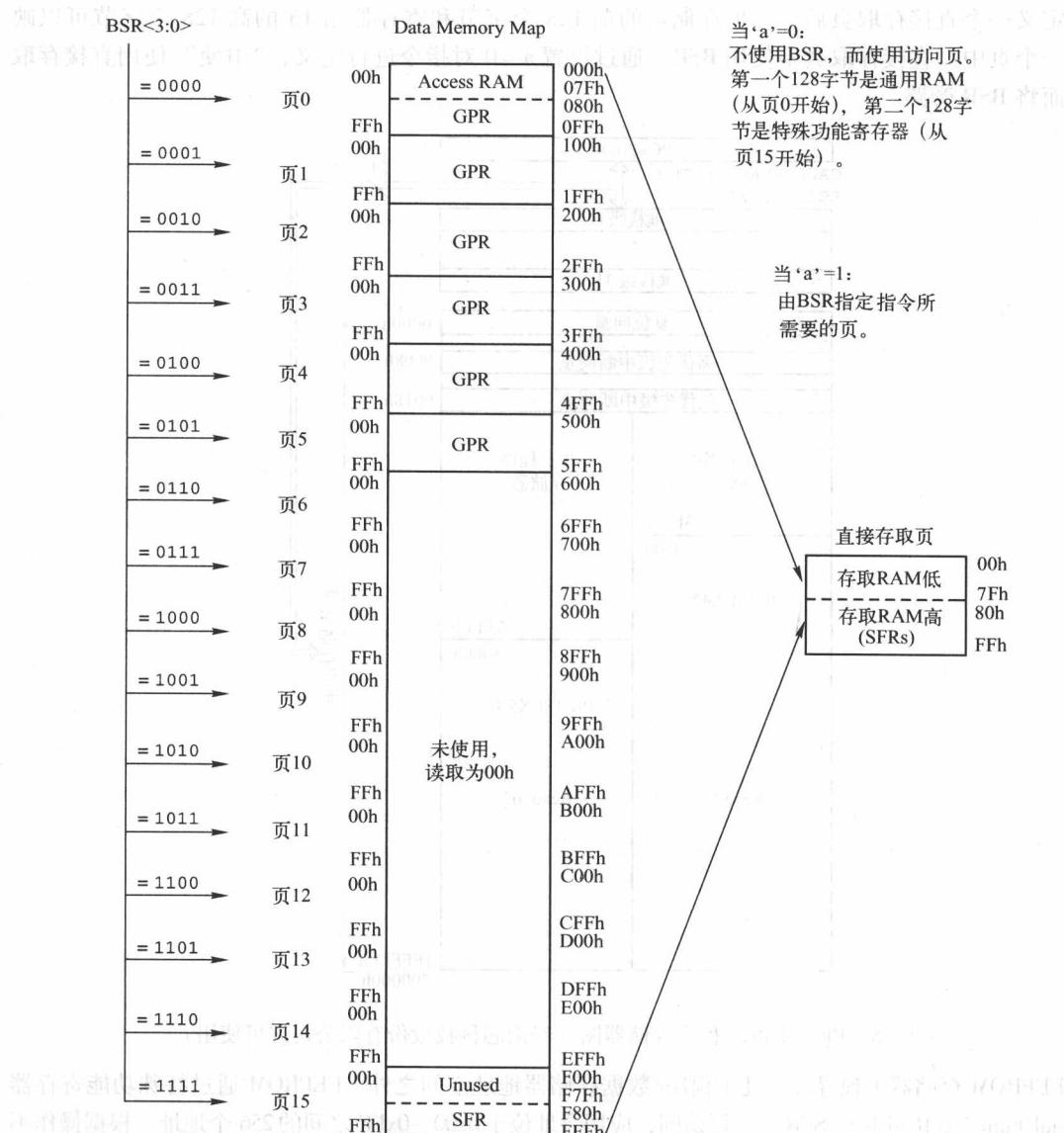


图 11-9 PIC18F4520 数据存储器图 (经微芯科技股份有限公司许可使用)

在中断期间, 返回的 PC 地址将被保存在堆栈中。此外, 工作寄存器 (working register, WREG)、Status 和 BSR 寄存器也被保存在快速返回堆栈中。如果未使用从中断的快速返回功能, 用户需要将 WREG、Status 和 BSR 寄存器保存在中断服务子程序的入口处。另外, 根据应用的不同, 其他寄存器可能也需要保存。

### 11.1.9 输入/输出 (I/O) 端口

PIC18F4520 中定义了 5 个 I/O 端口, 大多数引脚都与外设功能共用, 以减少整个芯片封装的管脚数量。一般来说, 除非用作外设功能, 所有的管脚都可以用于按位 I/O 操作。图 11-11 展示了通用 I/O 端口的结构图。

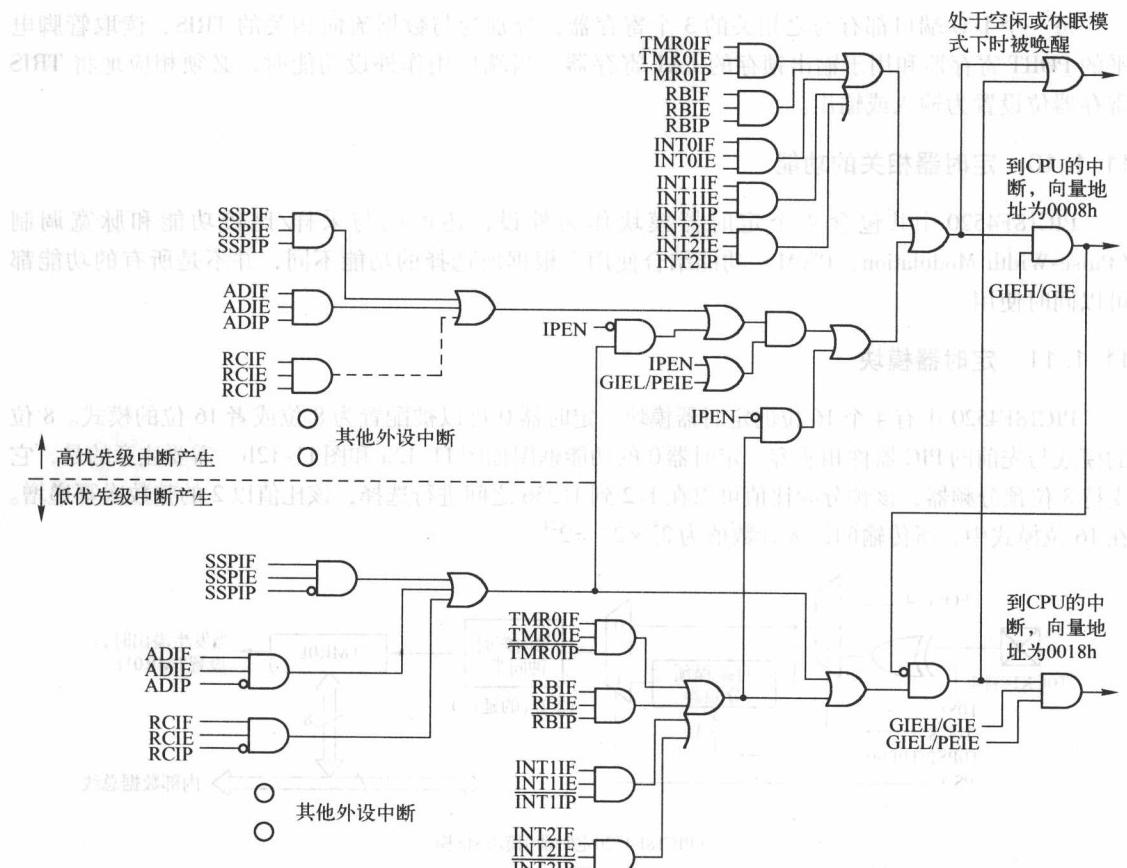
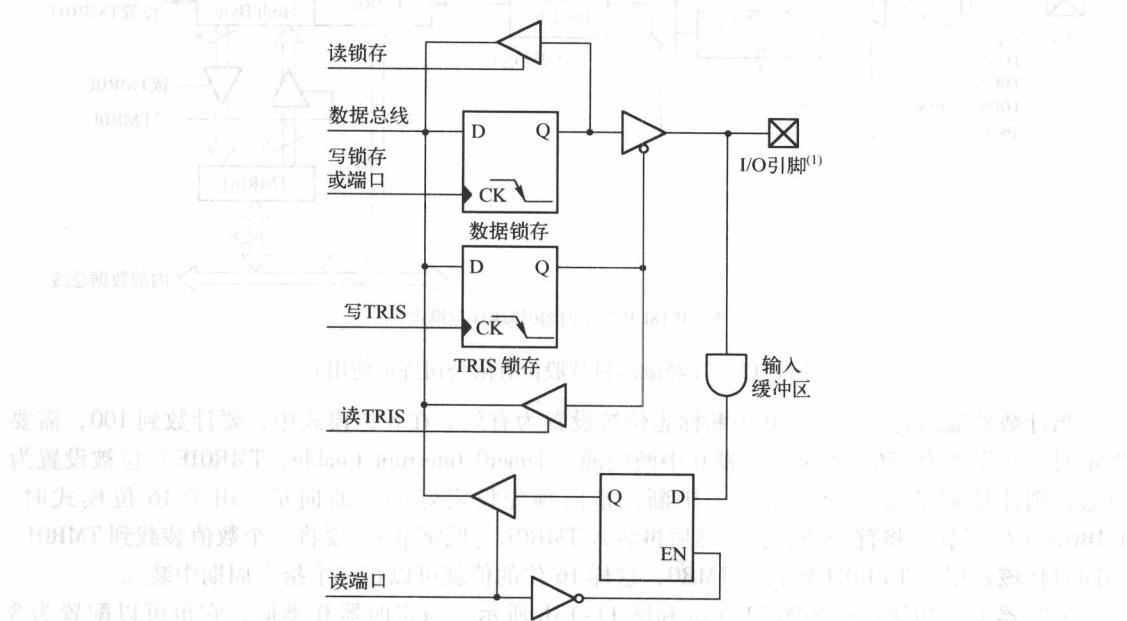


图 11-10 PIC18F4520 中断结构 (经微芯科技股份有限公司许可使用)



注：I/O 引脚需要经过二极管保护连接到 V<sub>DD</sub> 和 V<sub>SS</sub>。

图 11-11 PIC18F4520 通用 I/O 端口 (经微芯科技股份有限公司许可使用)

每一个 I/O 端口都有与之相关的 3 个寄存器，分别为与数据流向相关的 TRIS、读取管脚电平的 PORT 寄存器和用于输出锁存的 LAT 寄存器。当端口用作外设功能时，必须相应地将 TRIS 寄存器位设置为输入或输出。

### 11.1.10 定时器相关功能

PIC18F4520 中共包含 4 个定时器模块作为外设，还可以与采样/比较功能和脉宽调制（Pulse-Width Modulation，PWM）功能结合使用。根据所选择的功能不同，并不是所有的功能都可以同时使用。

### 11.1.11 定时器模块

PIC18F4520 中有 4 个 16 位的定时器模块。定时器 0 可以被配置为 8 位或者 16 位的模式。8 位的模式与先前的 PIC 器件相兼容。定时器 0 的功能框图如图 11-12a 和图 11-12b。需要注意的是，它支持 8 位预分频器，该预分频比值可以在 1:2 到 1:256 之间进行选择，该比值以 2 的整数次幂递增。在 16 位模式中，所传输的最大计数值为  $2^8 \times 2^{16} = 2^{24}$ 。

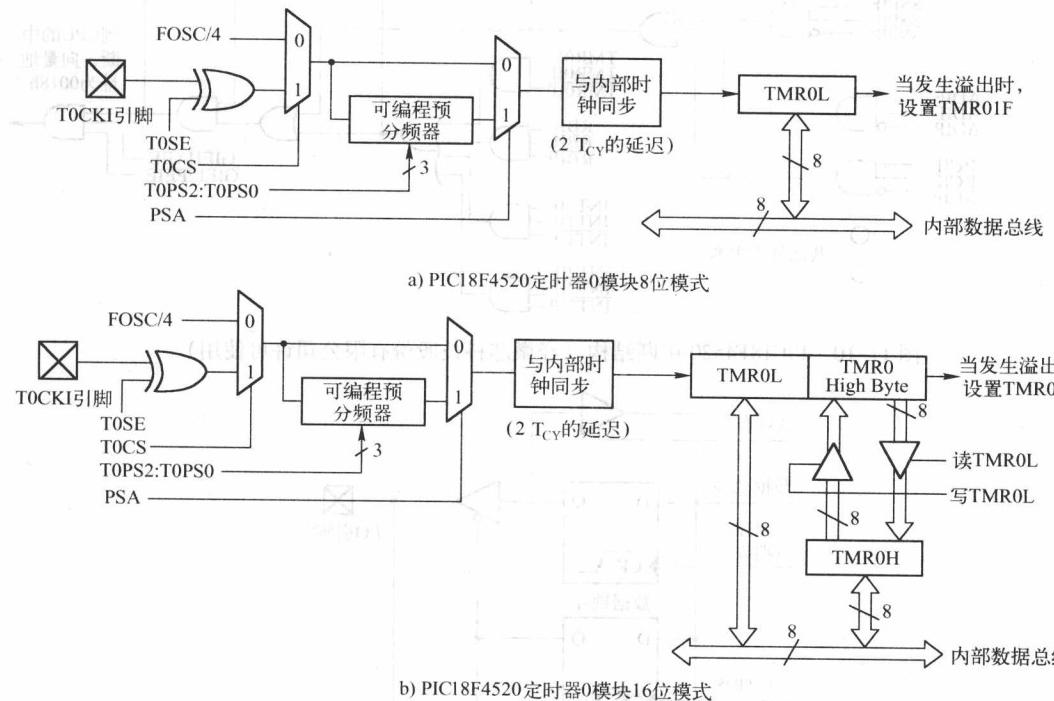


图 11-12 (经微芯科技股份有限公司许可使用)

当计数器溢出时，定时器 0 中断标志位被设置为有效。在 8 位模式中，要计数到 100，需要将定时器 0 设置为 156。如果定时器 0 中断使能（Timer0 Interrupt Enable, TMR0IE）位被设置为有效，则计数器的溢出会产生一个中断，指向预先设定好的中断向量。用于 16 位模式时，TMR0H（高字节）将首先被装入，然后再装入 TMROL（低字节）。要将一个数值装载到 TMROL，会同时将该数据从 TMROH 装载到 TMRO，这样 16 位的值就可以在一个指令周期中装入。

定时器 1 的功能框图如图 11-13a 和图 11-13b 所示，与定时器 0 类似，它也可以配置为 8 位或者 16 位的操作模式，但是它的预分频器选择限制比较多。定时器 1 共支持如下 3 种主要的操作模式：

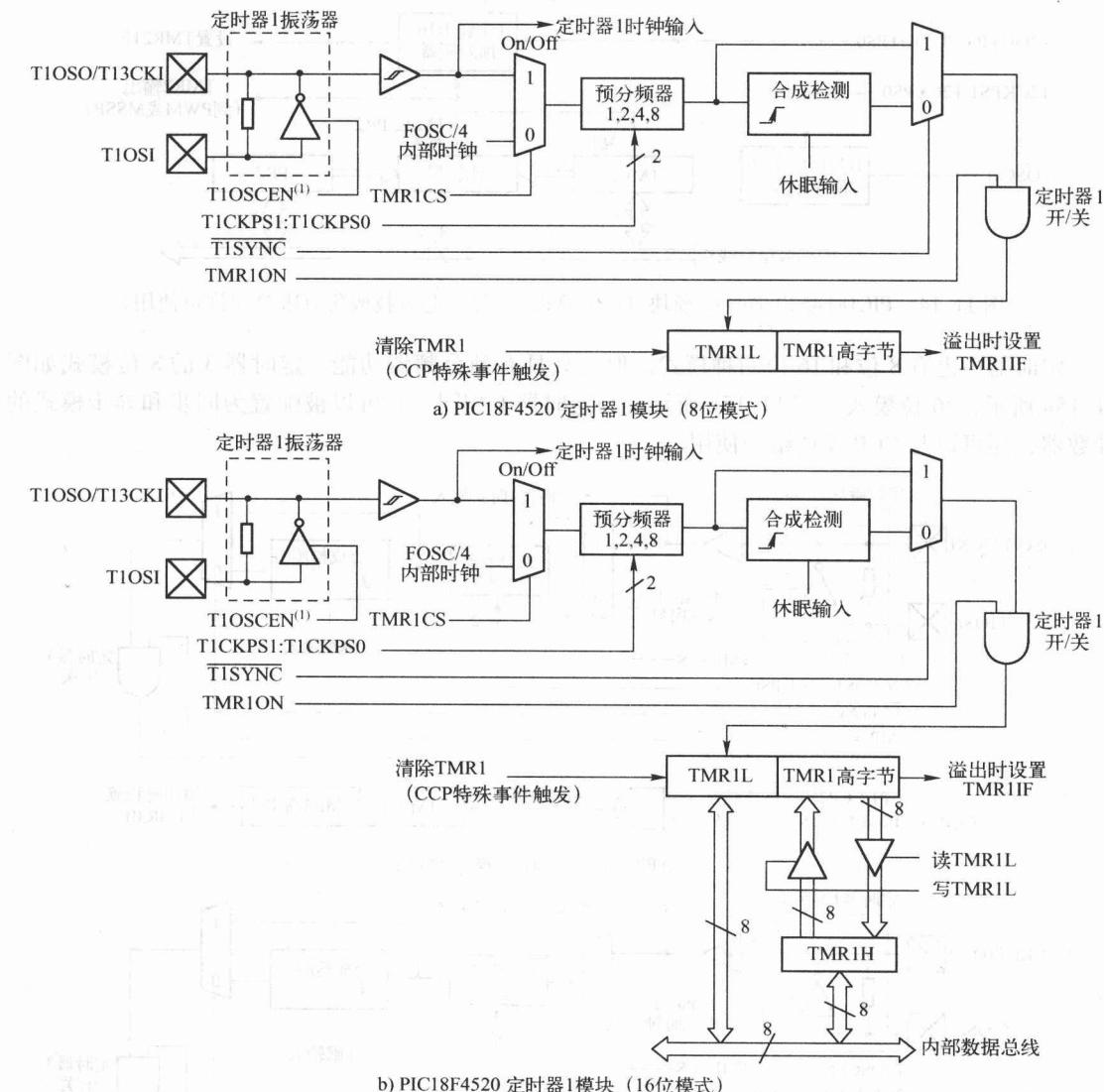


图 11-13 (经微芯科技股份有限公司许可使用)

- 定时器。
- 同步计数器。
- 异步计数器。

可以配置定时器 1，支持外部振荡器。这个功能非常有用，可以在电源管理模式中将定时器 1 用作一个时钟源。另外，定时器 1 还可以用作实时时钟。如果定时器 1 使用的振荡器频率为 32.768 kHz，并工作在 16 位的读/写模式下，溢出时间将为 2 s。适当设置 TMR1H 和 TMR1L 寄存器对，可缩短溢出时间。

8 位模式定时器 2，具有预分频器和后分频器功能，如图 11-14 所示。它同时包含了预分频器和后分频器功能，可以将 TMR2 寄存器的值与周期寄存器（Period Register, PR）的值进行比较，并将结果输出到 PWM 或者主控同步串行端口（Master Synchronous Serial Port, MSSP）模块。

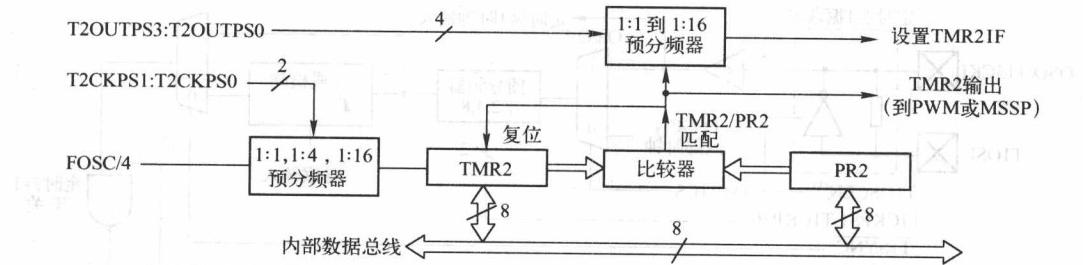


图 11-14 PIC18F4520 Timer2 模块 (8 位模式) (经微芯科技股份有限公司许可使用)

定时器 3 也有 8 位和 16 位两种模式，但是只具有预分频器功能。定时器 3 的 8 位模式如图 11-15a 所示，16 位模式如图 11-15b 所示。与定时器 1 相同，它可以被配置为同步和异步模式的计数器，还可以与 CCP 模块组合使用。

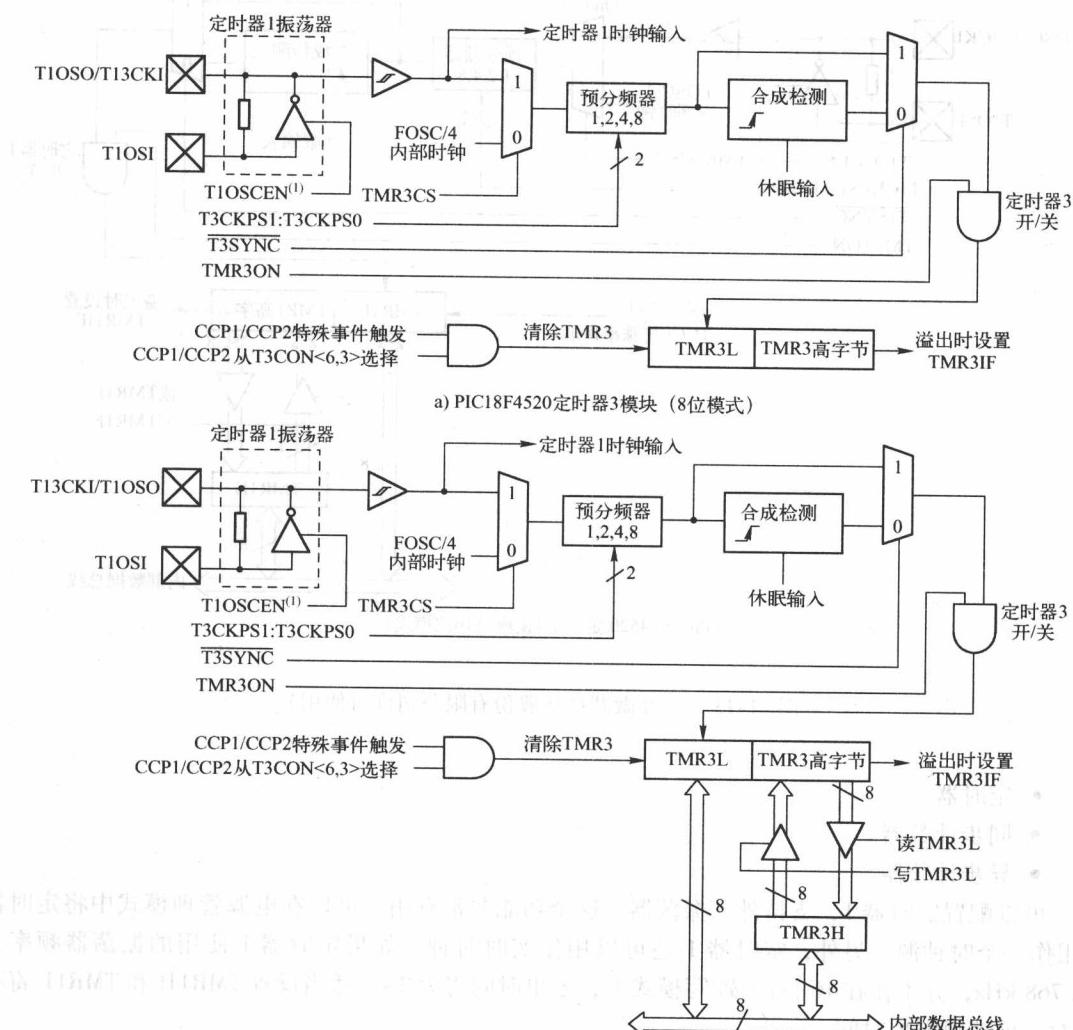


图 11-15 (经微芯科技股份有限公司许可使用)

### 11.1.12 采样/比较/PWM 功能

有三种特殊的功能需要使用定时器，分别是采样（见图 11-16）、比较（见图 11-17）和脉宽调制（见图 11-18）。在同一时刻不可能所有的定时器都有效，这样就限制了在一个设计中可以包含的功能的数量，这取决于选择什么样的组合来用作定时器、CCP 或者 PWM 功能。

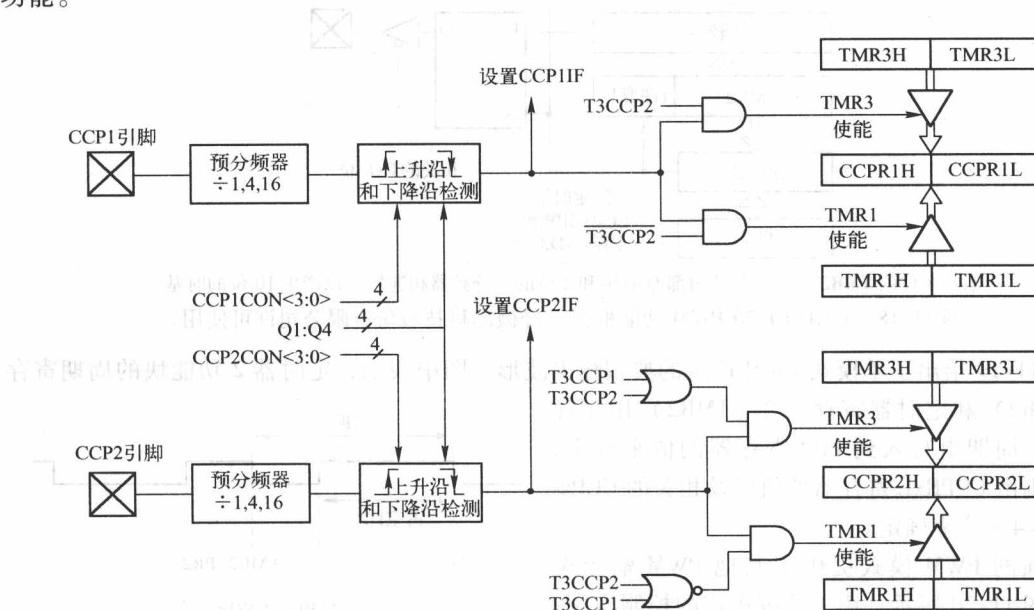


图 11-16 PIC18F4520 采样功能框图（经微芯科技股份有限公司许可使用）

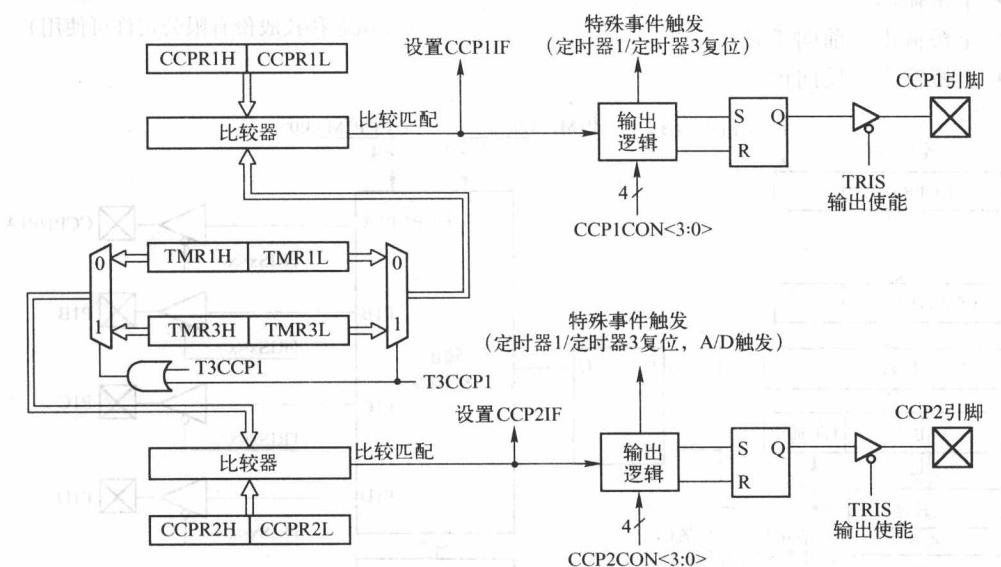
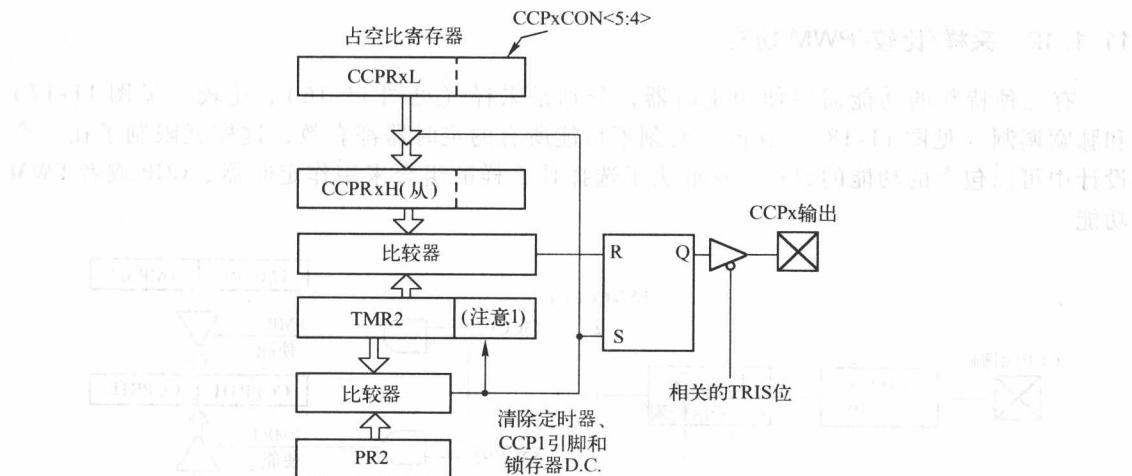


图 11-17 PIC18F4520 比较功能框图（经微芯科技股份有限公司许可使用）

在普通模式和增强模式中都支持脉宽调制功能。图 11-18 给出了 PWM 功能的简单框图。它支持多达 10 位的分辨率。这种标准的 PWM 功能仅在单输出模式中可以使用。



注：8 位的 TMR2 值与 2 位的内部 Q 时钟和 2 位的预分频器相连接，以产生 10 位的时基。

图 11-18 PIC18F4520 PWM 功能框图（经微芯科技股份有限公司许可使用）

图 11-19 给出了单模式 PWM 产生的典型输出波形。图中表示，定时器 2 功能块的周期寄存器 2 (PR2) 和定时器寄存器 2 (TMR2) 用于计算波形。周期由写入到 PR2 寄存器的值来确定，占空比则由 CCPRxL 寄存器的值以及相关的 CCPx-CON <5:4> 位来确定。

增强的 PWM 模式提供了其他 PWM 输出选项，如图 11-20 所示，用于范围更广的控制应用。它支持 4 个输出，共有如下 3 种操作模式：

- 半桥输出。
- 全桥输出，前向模式。
- 全桥输出，反向模式。

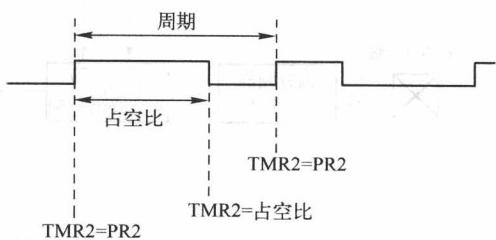


图 11-19 PIC18F4520 PWM 输出  
(经微芯科技股份有限公司许可使用)

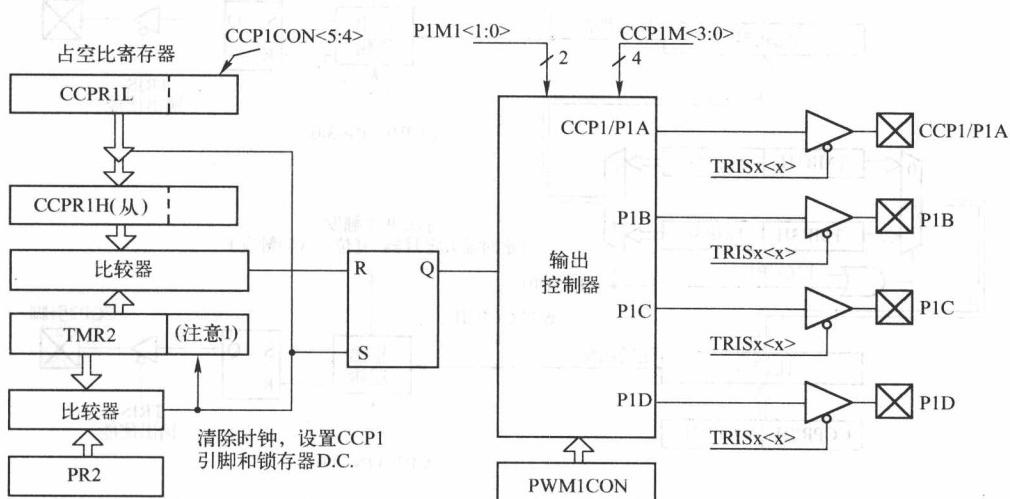


图 11-20 PIC18F4520 增强的 PWM 框图（经微芯科技股份有限公司许可使用）

在半桥输出模式中，两个管脚用作输出，驱动推挽负载。PWM 的输出信号在 P1A 引脚输

出，而 PWM 的互补输出信号在 P1B 引脚输出。这种模式可以被用作半桥或者全桥应用，如图 11-21a 和图 11-21b 所示。

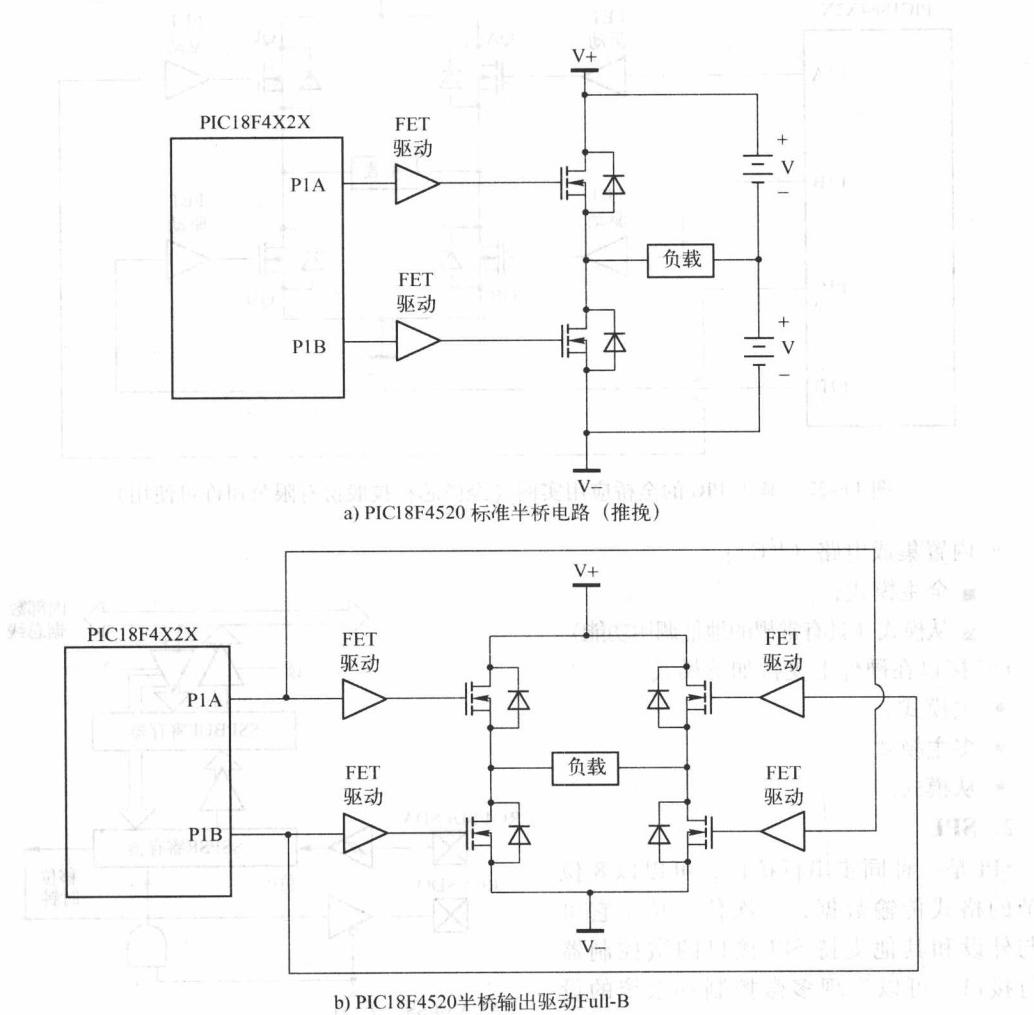


图 11-21 (经微芯科技股份有限公司许可使用)

全桥输出模式中有 4 个引脚用作输出，但是在同一时刻只有 2 个输出是有效的。前向模式中，P1A 引脚是一直有效的，P1D 被调制。反向模式中，P1C 引脚是一直有效的，P1B 被调制。如图 11-22 所示为全桥模式的一种应用。

### 11.1.13 串行通信接口

PIC18F4520 主要支持 4 种串行 I/O 功能：具有 SPI 和 I<sup>2</sup>C 模式的 MSSP 以及增强的 USART，还支持 LIN 1.2 功能。这样就对那些需要高性能串行 I/O 功能的应用提供了更广泛的支持。

#### 1. MSSP

MSSP 模块是一个串行接口，在与其他外设或者微控制设备进行通信时非常有用。例如，这些设备可以是 EEPROM、移位寄存器、显示驱动器或者 A/D 转换器（ADC）。MSSP 模块可以工作于如下两种模式之一：

- 串行外设接口（SPI）。

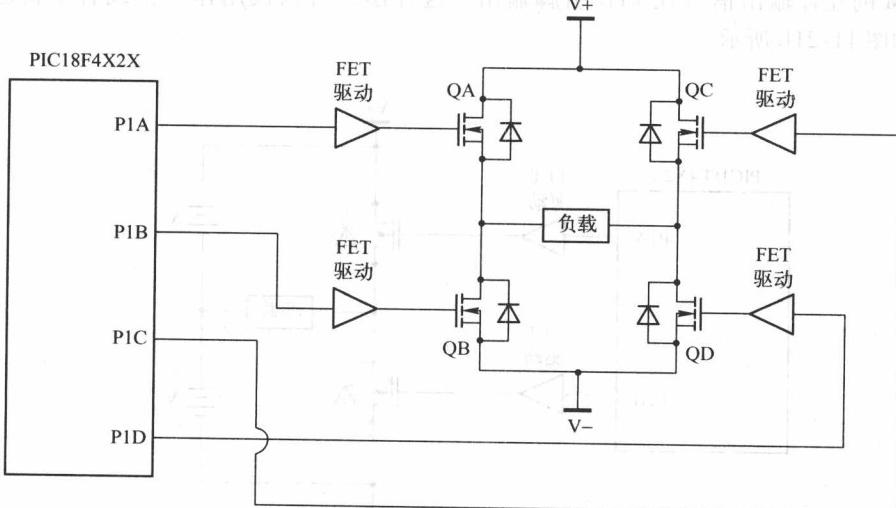


图 11-22 基于 PIC 的全桥应用实例（经微芯科技股份有限公司许可使用）

- 内置集成电路 ( $I^2C$ )：

- 全主模式；
- 从模式（具有常规的地址调用功能）。

$I^2C$  接口在硬件上支持如下模式：

- 主模式。
- 多主模式。
- 从模式。

## 2. SPI

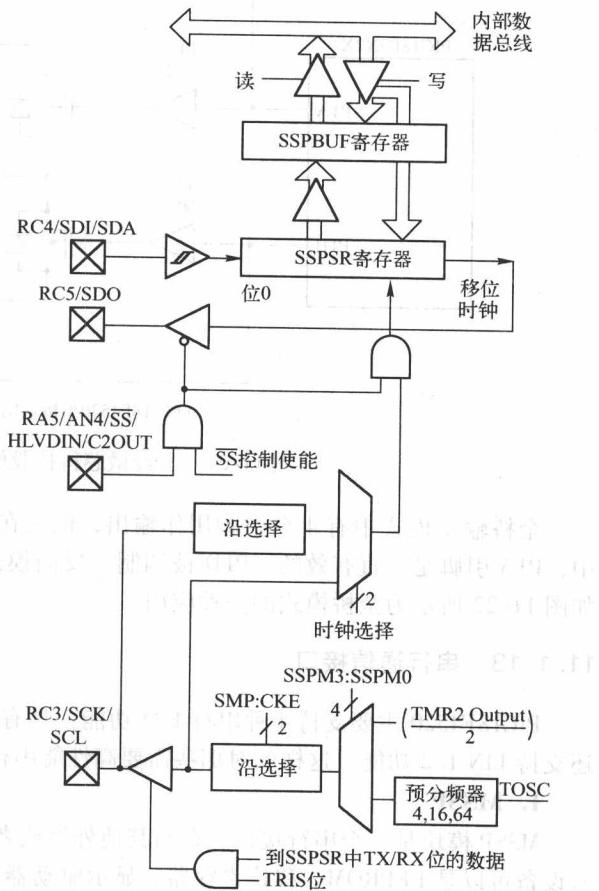
SPI 是一种同步串行接口，可以以 8 位字节的格式传输数据，一次传一位。它可以与外设和其他支持 SPI 接口的微控制器进行接口，可以实现多微控制器系统的设计。图 11-23 给出了基本的 SPI 接口逻辑。

两个微控制器之间的典型连接如图 11-24 所示。通过发送串行时钟 (Serial Clock, SCK) 信号，主控制器发起数据传输过程。数据在由编程指定的时钟沿输出到移位寄存器中，并在相反的时钟沿锁存。

在主模式中，主设备可以随时发起数据传输过程，因为它控制着 SCK。通过软件协议的设定，主设备确定了从设备（见图 11-24 中的处理器 2）何时传播数据。在从模式中，当外部时钟脉冲出现在 SCK 时，进行数据的发送和接收。

## 3. $I^2C$

$I^2C$  功能图如图 11-25 所示，处于  $I^2C$  模

图 11-23 PIC 18F4520 MSSP 框图 (SPI 模式)  
(经微芯科技股份有限公司许可使用)

式下的 MSSP 完全实现所有的主设备和从设备功能（包括一般的调用支持），并在硬件的起始位和结束位上提供中断，以定义空闲总线（多主设备功能）。MSSP 模块可实现标准的模式，也实现 7 位和 10 位寻址方式，SCK 和串行数据（Serial Data，SDA）管脚通过 RC3 和 RC4 引脚进行配置。

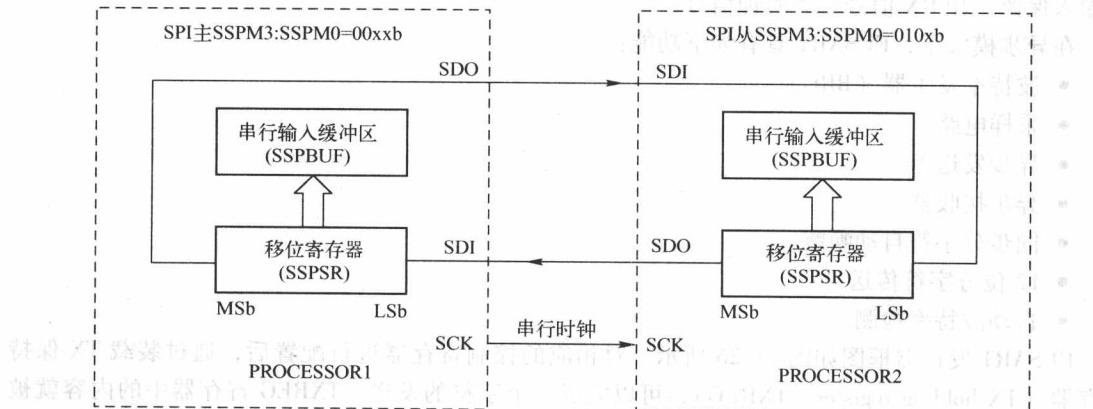


图 11-24 PIC18F4520 SPI 主/从连接（经微芯科技股份有限公司许可使用）

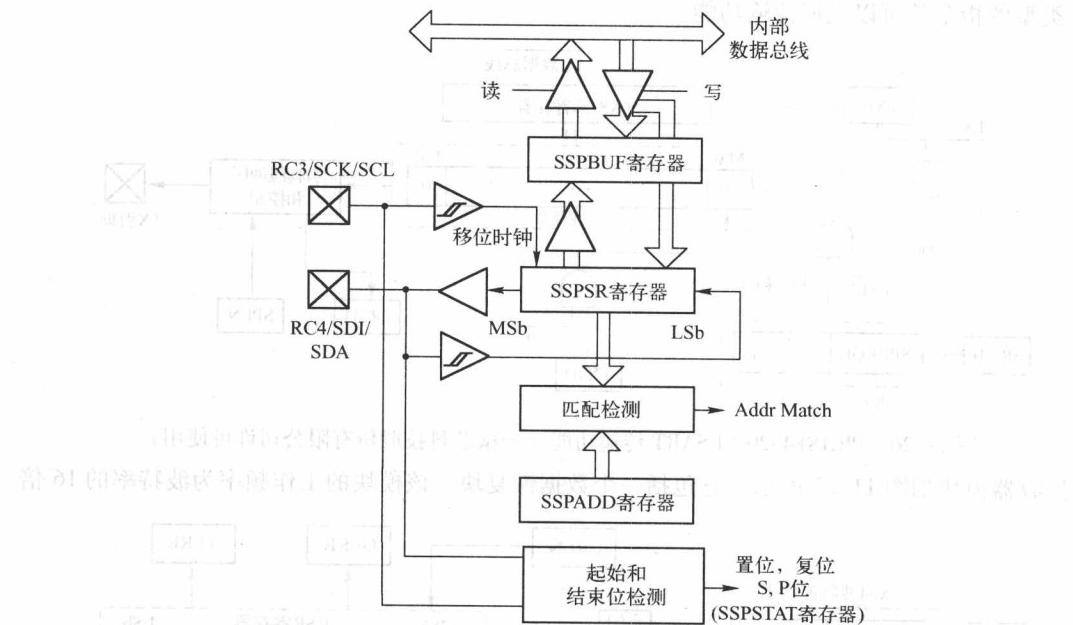


图 11-25 PIC18F4520 I<sup>2</sup>C 功能图（经微芯科技股份有限公司许可使用）

#### 4. EUSART

PIC18F4520 支持增强的 USART 模块，通常被称为串行通信接口或者 SCI。EUSART 可以配置为全双工异步系统与外设进行通信，例如 CRT 终端和个人计算机；也可以配置为半双工异步系统与外设进行通信，例如 A/D 或者 D/A 转换器和串行 EEPROM。

总体来讲，EUSART 可以配置为如下模式。

- 具有以下特点的异步（全双工）模式：
  - 接收字符自动唤醒；
  - 自动波特校准；
  - 12 位的分字符传送。

- 同步模式——具有可选择时钟精度的主设备（半双工）。
- 同步模式——具有可选择时钟精度的从设备（半双工）。

EUSART 可以在接收时自动检测波特率，波特率发生器（Baud Rate Generator, BRG）的时钟输入保留，让 RX 信号作为时钟信号。

在异步模式下，EUSART 具有如下功能：

- 波特率发生器（BRG）。
- 采样电路。
- 异步发送器。
- 异步接收器。
- 同步分字符自动唤醒。
- 12 位分字符传送。
- 自动波特率检测。

EUSART 发送器框图如图 11-26 所示。对相应的控制寄存器进行配置后，通过装载 TX 保持寄存器（TX holding register, TXREG），可以完成一个字符的发送。TXREG 寄存器中的内容就被传送到 TSR 中，也就是它应该被传送到的地方。因此，一旦控制寄存器的值被设定后，简单的 MOV 类型的指令就可以完成传输功能。

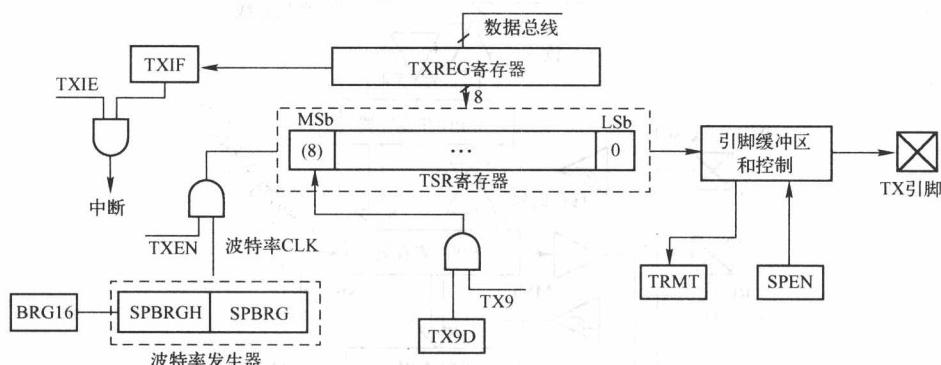


图 11-26 PIC18F4520 EUSART 传输功能（经微芯科技股份有限公司许可使用）

接收器模块如图 11-27 所示。它包括一个数据恢复块，该模块的工作频率为波特率的 16 倍。

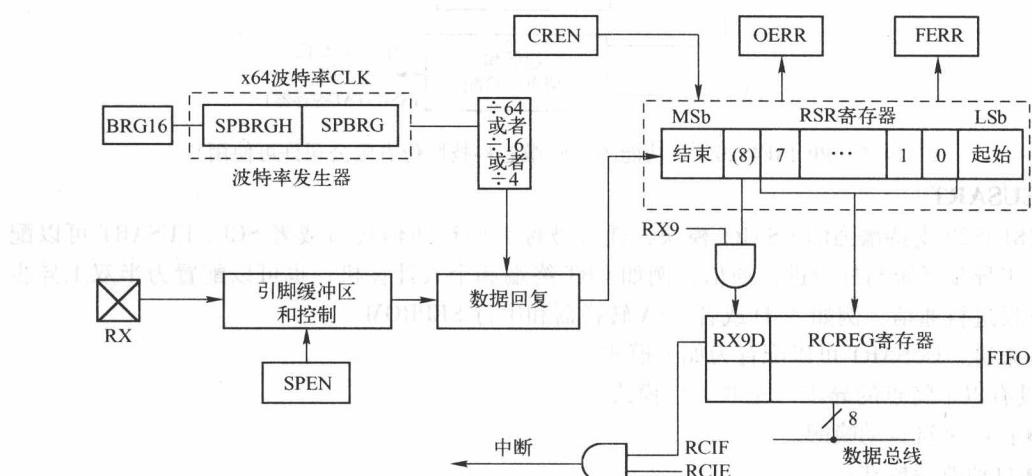


图 11-27 PIC18F4520 接收器框图（经微芯科技股份有限公司许可使用）

对于RS-232接收，其接收串行移位器的工作频率为晶振频率(Fosc)。

通过中断或查询完成可以接收功能。例如，通过查询操作可以检查接收标志位(receive flag, RCIF)，而接收的字节只需要从RCREG移动(MOV)到文件的适当位置即可。

### 11.1.14 模数转换

ADC功能框图如图11-28所示，对于较小的28管脚芯片，有10个输入；对于40/44管脚的芯片，有13个输入。该模块可以将一个模拟输入信号转换为相应的10位数字信号。输出数据是通过不断近似的方法获得的。

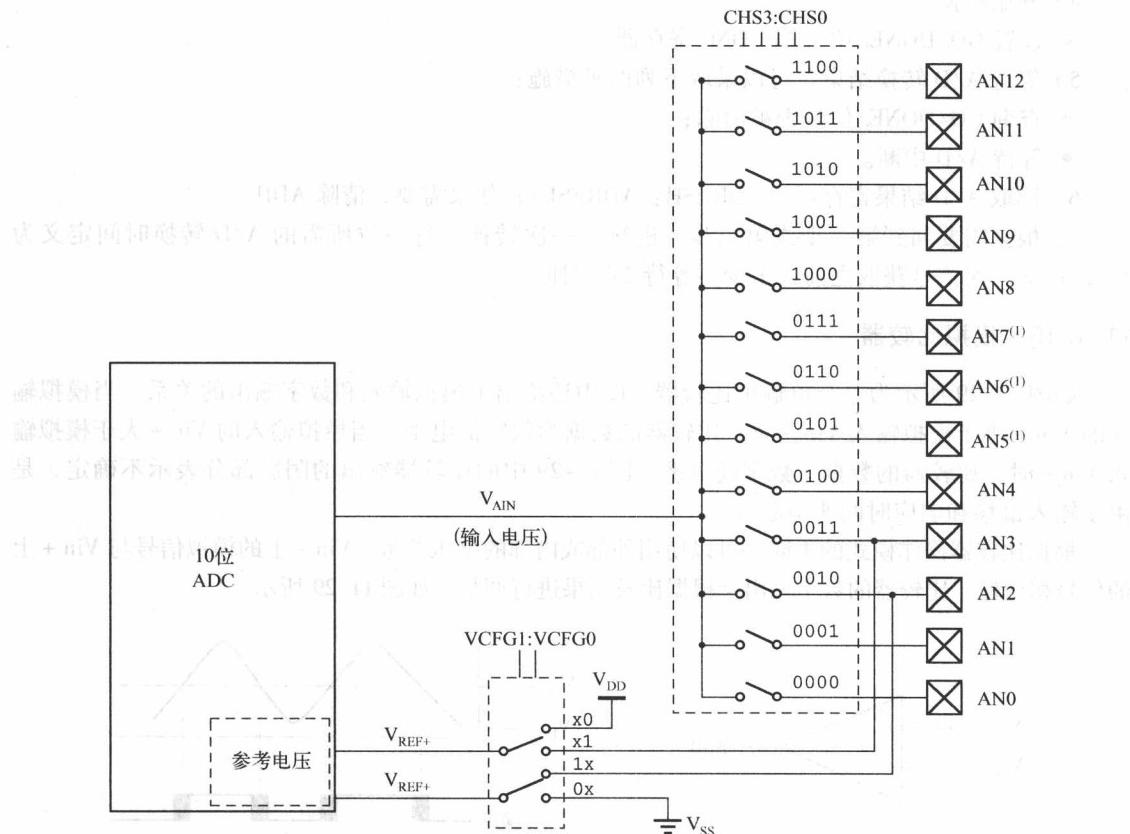


图11-28 PIC18F4520 ADC框图（经微芯科技股份有限公司许可使用）

模拟参考电压可以通过软件进行选择，可以是设备的正电压或负电压(V<sub>dd</sub>和V<sub>ss</sub>)，也可以是RA3/AN3/参考电压+和RA2/AN2/参考电压-/C参考电压引脚的电压。ADC还有一个特点，即可以工作在休眠模式下。要想使其工作在休眠模式下，A/D转换时钟必须由ADC的内部RC振荡器驱动。

通过对ADCON0、ADCON1、ADCON0控制寄存器的相应位进行适当设置，可以使用A/D转换功能。一次转换过程可以由如下步骤组成：

#### 1) 配置A/D模块：

- 配置模拟管脚、参考电压和数字I/O(ADCON1)；
- 选择A/D输入通道(ADCON0)；
- 选择A/D输入通道(ADCON0)；

- 选择 A/D 获取时间 (ADCON2);
  - 选择 A/D 转换时钟 (ADCON2);
  - 激活 A/D 模块 (ADCON0)。
- 2) 配置 A/D 中断 (如果需要的话):
- 清除 ADIF 位;
  - 置位 ADIE 位;
  - 置位 GIE 位。
- 3) 等待需要的获取时间 (如果需要的话)。
- 4) 开始转换:
- 设置 GO/DONE/位 (ADCON0 寄存器)。
- 5) 等待 A/D 转换结束, 可以采取下列两种措施:
- 查询 GO/DONE/位是否被清除;
  - 等待 A/D 中断。
- 6) 读取 A/D 结果寄存器 (ADRESH: ADRESL); 如果需要, 清除 ADIF。
- 7) 根据需要回到第一步或第二步, 进行下一次转换。每一位所需的 A/D 转换时间为  $T_{AD}$ , 在下一次信号获取之前, 至少要等待  $2T_{AD}$  时间。

### 11.1.15 模拟比较器

如图 11-29 所示为一个单输出比较器, 图中还给出了模拟输入和数字输出的关系。当模拟输入的  $V_{IN+}$  小于模拟输入  $V_{IN-}$  时, 比较器的数据为数字低电平。当模拟输入的  $V_{IN+}$  大于模拟输入  $V_{IN-}$  时, 比较器的数据为数字高电平。图 11-29 中的比较器输出的阴影部分表示不确定, 是由于输入偏移和响应时间所引起的。

根据比较器操作模式的不同, 可以使用外部或内部的电压参考。 $V_{IN-}$  上的模拟信号与  $V_{IN+}$  上的信号相比较, 比较器的数字输出会根据比较结果进行调整, 如图 11-29 所示。

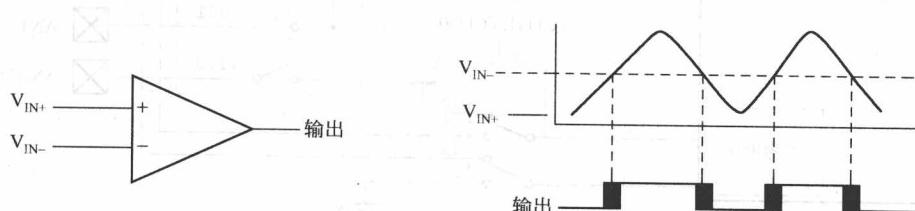


图 11-29 PIC18F4520 单输出比较器 (经微芯科技股份有限公司许可使用)

比较器的框图如图 11-30 所示, 其输出可以通过 CMCON 寄存器读取。寄存器中的数据是只读的。比较器的输出也可以直接输出到 RA4 和 RA5 I/O 管脚, 如果设置为使能, 位于 RA4 和 RA5 引脚输出路径中的多路器将进行切换, 这样每一个管脚的输出将是比较器结果的非同步输出。每一个比较器的不确定性与输入电压的偏移和响应时间有关, 这些数据都会在说明书中给出。

### 11.1.16 CPU 特性

PIC18F4520 中引入了许多特性, 目的是通过减小外部组成设备, 提高可靠性, 降低成本。这些特性如下所述:

- 振荡器选择。
- 复位:

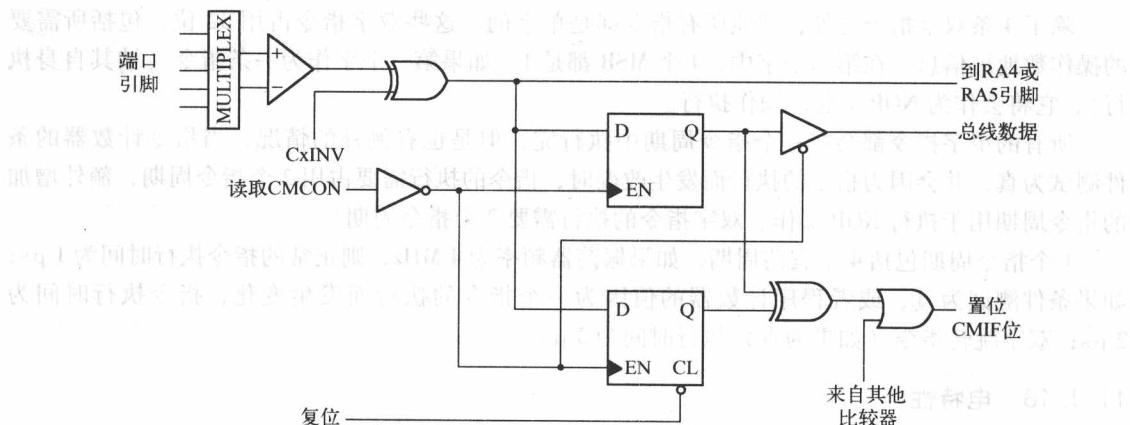


图 11-30 PIC18F4520 比较器框图（经微芯科技股份有限公司许可使用）

- 上电复位 (POR)；
- 加电定时器 (Power-up Timer, PWRT)；
- 振荡器启动定时器 (Oscillator Start-up Timer, OST)；
- 欠压复位 (BOR)。
- 中断。
- Watchdog 定时器 (WDT)。
- 故障保护时钟监视器。
- 双速启动。
- 代码保护。
- ID 定位。
- 在线串行编程。

根据频率、功耗、精度和成本，用户可以对应用的振荡器进行配置。除了为复位提供的加电和振荡器启动定时器之外，PIC18F4520 还有一个 Watchdog 定时器，该定时器可以配置为永久启用，也可以由软件进行控制（适用于不能进行配置的情况）。

### 11.1.17 指令集

PIC18F4520 实现的是精简指令集体系结构 (RISC)，大多数指令（非跳转指令）可以在一个指令周期中执行完。PIC18F4520 包含 75 条 PIC 18 内核指令的标准集合，还包含了 8 条扩展的指令，这 8 条指令用于优化递归的指令代码，或者使用软件堆栈实现。

标准的 PIC18 指令集在 PICmicro 指令集的基础上有了许多改进，而同时又保持了 PICmicro 指令集的移植性；大多数指令占用一个程序存储器字（16 位），但有 4 条指令需要两个程序存储器字，分别是 MOVFF、CALL、GOTO 和 LSFR。

指令集中，每一条单字指令都是 16 位的，分为操作码和一个或两个操作数，分别用于说明指令的类型，并进一步说明指令的操作。

指令集高度正交，可以分为如下 4 种类型：

- 面向字节的操作。
- 面向位的操作。
- 控制操作。
- 字符操作。

除了 4 条双字指令之外，其他所有指令都是单字的。这些双字指令占用 32 位，包括所需要的操作数地址信息。在第二个字中，4 个 MSB 都是 1。如果第二个字作为一条指令（被其自身执行），它将会作为 NOP（空）操作执行。

所有的单字指令都会在一个指令周期中执行完，但是也有例外的情况，当指令计数器的条件测试为真，并会因为指令的执行而发生改变时，指令的执行需要占用 2 个指令周期，额外增加的指令周期用于执行 NOP 操作。双字指令的执行需要 2 个指令周期。

1 个指令周期包括 4 个震荡周期，如果振荡器频率为 4 MHz，则正常的指令执行时间为 1  $\mu$ s；如果条件测试为真，或者程序计数器的值因为一个指令的执行而发生变化，指令执行时间为 2  $\mu$ s；双字跳转指令（如果为真）执行时间为 3  $\mu$ s。

### 11.1.18 电特性

主要的电特性如图 11-31 所示。注意，所有端口同时输出或输入的总电流是受限制的。这些值在设计过程中是需要考虑的限制因素。

绝对最高等级	
环境温度下的偏差	-40°C ~ +125°C
存储温度	-65°C ~ +150°C
任一引脚相对于 VSS 的电压（除 VDD、MCLR 和 RA4 之外）	-0.3V ~ (VDD+0.3V)
VDD 相对于 VSS 的电压	-0.3 ~ +7.5V
MCLR 相对于 VSS 的电压（注意2）	0 ~ +13.25V
总功耗（注意1）	1.0W
VSS 引脚的最大电流切削	300mA
VDD 引脚的最大输入电流	250mA
输入阻尼电流，I <sub>lk</sub> ( $V_i < 0$ 或 $V_i > VDD$ )	$\pm 20$ mA
输出阻尼电流，I <sub>ok</sub> ( $V_o < 0$ 或 $V_o > VDD$ )	$\pm 20$ mA
任何 I/O 引脚的最大输出反向电流	25 mA
任何 I/O 引脚的最大输入反向电流	25 mA
所有端口能够吸收的最大电流	200 mA
所有端口能够发出的最大电流	200 mA

图 11-31 PIC18F4520 电特性（经微芯科技股份有限公司许可使用）

## 11.2 ZiLOG Z8 ENCORE! XP F0830 系列

ZiLOG Z8 ENCORE! XP F0830 系列控制器是基于 ZiLOG eZ8 CPU 的 Flash 微控制器。MCU 是 Z8 ENCORE! XP 系列器件的一部分，其指令集结构与最初的 Z8 设计有很多相似之处。

ZiLOG Z8 ENCORE! XP F0830 系列是通用的 8 位微控制器。由于具有灵活的硬件设计特性，其可以以较低的成本用于大量的应用。它支持一种基本的流水线体系结构，针对多字节、多周期指令，对取指和执行进行了优化。

图 11-32 给出了 Z8 ENCORE! XP F0830 系列控制器框图。它最多可以支持 12 KB 的 Flash 程序存储器和 256 字节的寄存器 RAM，还包含诸多外设功能，如 PWM 和 WDT 以及最多 8 通道的具有 SAR 转换器的快速模数转换 (11.9  $\mu$ s)。

Z8 ENCORE! XP F0830 系列 MCU 具有如下主要特性：

- 20 MHz eZ8 CPU。
- 最多 12 KB 的 Flash 存储器，具有现场编程功能。
- 最多 25B 的寄存器 RAM。
- 64B 非易失性数据存储器 (Nonvolatile data storage, NVDS)。

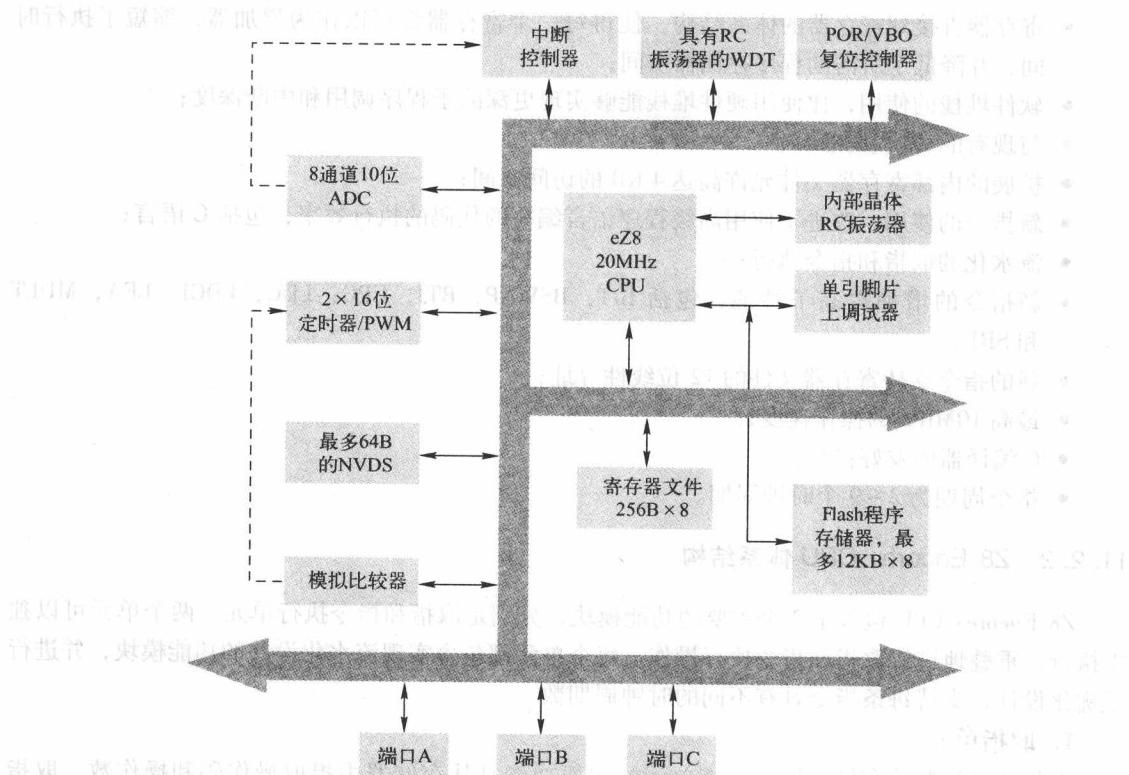


图 11-32 | Z8 ENCORE! XP F0830 系列框图 (经 ZiLOG 公司许可使用)

- 根据封装不同，具有最多 25 个 I/O 引脚。
- 内部精密振荡器 (Internal precision oscillator, IPO)。
- 外部晶体振荡器。
- 2 个增强的 16 位定时器，具有采样和比较功能。
- Watchdog 定时器 (WDT)，具有专用的内部 RC 振荡器。
- 单引脚片上调试器 (OCD)。
- 可选的 8 通道 1 位模数转换器 (ADC)。
- 片上模拟比较器。
- 最多 17 个中断源。
- 欠电压保护 (Voltage brownout protection, VBO)。
- 上电复位 (Power-on reset, POR)。
- 2.7 V ~ 3.6 V 工作电压。
- 最多 13 个 5V 输入引脚。
- 20 引脚和 28 引脚封装。

### 11.2.1 eZ8 CPU 描述

ZiLOG Z8 ENCORE! XP F0830 eZ8 CPU 是一种速度更快、面向代码的微控制器，它对 Z8 指令集进行了改进，使得指令格式更加优化；指令的字节数也很小，确保代码的紧凑性。这一点还使得指令的执行速度加快，因为在执行过程中减小了指令字节流的数量。

eZ8 CPU 有如下主要特性：

- 寄存器直接到寄存器的体系结构，使得每一个寄存器都可以作为累加器，缩短了执行时间，并降低了所需的程序存储器空间；
- 软件堆栈的使用，比使用硬件堆栈能够实现更深的子程序调用和中断深度；
- 与现有的 Z8 代码兼容；
- 扩展的内部寄存器文件允许高达 4 KB 的访问空间；
- 新指令的使用，改进了使用高级程序语言编写的代码的执行效率，包括 C 语言；
- 流水化的取指和指令执行；
- 新指令的增加提高了性能，包括 BIT, BSWAP, BTJ, CPC, LDC, LDCI, LEA, MULT 和 SRL；
- 新的指令支持寄存器文件的 12 位线性寻址；
- 最高 10MIPS 的操作速度；
- C 编译器的友好接口；
- 指令周期为 2 ~ 9 个时钟周期。

### 11.2.2 Z8 Encore! CPU 体系结构

Z8 Encore! CPU 包含了 2 个主要的功能模块，分别是取指和指令执行单元。两个单元可以独立执行，重叠地进行取指和指令执行操作。每个单元都包含实现流水化设计的功能模块，并进行了优化设计，支持每条指令具有不同的时钟周期数。

#### 1. 取指单元

取指单元控制着存储器接口。该单元的主要功能是从存储器中提取操作码和操作数。取指单元是流水化的，并半独立于 eZ8 CPU 的其他部分运行。它还可以取中断向量，或者对程序或数据存储器进行读和写操作。取指单元还对操作码进行部分译码，以确定下面的操作需要取几个字节的数据。取指单元的操作流程如下所示：

- 第一步：取操作码；
- 第二步：确定操作数的大小（字节数）；
- 第三步：取操作数；
- 第四步：将操作码和操作数输入到指令状态机中。

#### 2. 执行单元

执行单元被进一步划分为指令状态机、程序计数器、CPU 控制寄存器、算术逻辑单元 (ALU)。图 11-33 给出了执行单元的功能设计框图。

eZ8 CPU 执行单元由指令状态机进行控制。取指单元执行了初步的操作译码之后，指令状态机获得控制权，并完成指令的执行；执行寄存器读和写操作，并产生地址。指令周期时间会根据指令的不同而不同，对于给定的时钟速度，可以获得更高的性能。

标准 CPU 指令的最小指令执行时间是 2 个时钟周期（只有 BRK 指令在 1 个周期中执行完）。由于不同的指令所需要的字节数不同，所以指令之间会产生延迟周期。当下一个指令执行所占用的字节数超过当前指令执行所需要的时钟周期数时，就会增加延迟周期。例如，如果 eZ8 CPU 在执行一条双周期指令的同时，还要去取一条 3

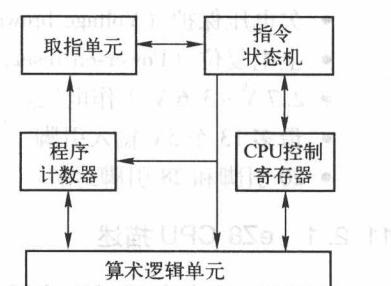


图 11-33 Z8 ENCORE! XP F0830  
执行单元框图（经 ZiLOG 公司许可使用）

个字节的指令，就会产生一个延迟周期，因为取指单元只有 2 个周期的时间来取 3 个字节。在延迟周期中，执行单元处于空闲状态。

程序计数器包含一个 16 位计数器和一个 16 位加法器。它显示当前存储器地址，并计算下一个存储器地址。它还会根据取指单元所取得的字节数，自动增加。16 位加法器用于增加程序计数器的值，对于相对寻址指令，还要控制程序计数器的跳转。

### 11.2.3 地址空间

eZ8 CPU 可以访问如下 3 个不同的地址空间：

- 寄存器文件包含通用寄存器以及 eZ8 CPU、外设和通用 I/O 端口控制寄存器的地址。
- 程序存储器包含所有保存可执行代码和/或数据的存储器地址。
- 数据存储器包含所有只保存数据的存储器地址。

#### 1. 寄存器文件

Z8 ENCORE! MCU 中的寄存器文件地址空间为 4 KB (4096 个字节)，如图 11-34 所示。寄存器文件由两部分组成，即控制寄存器和通用寄存器。执行指令时，从源寄存器中读取数据，并向目的寄存器中写入数据。eZ8 CPU 的体系结构允许所有的通用寄存器以正交的方式作为累加器、地址指针、索引寄存器、堆栈空间或者补丁存储器使用。

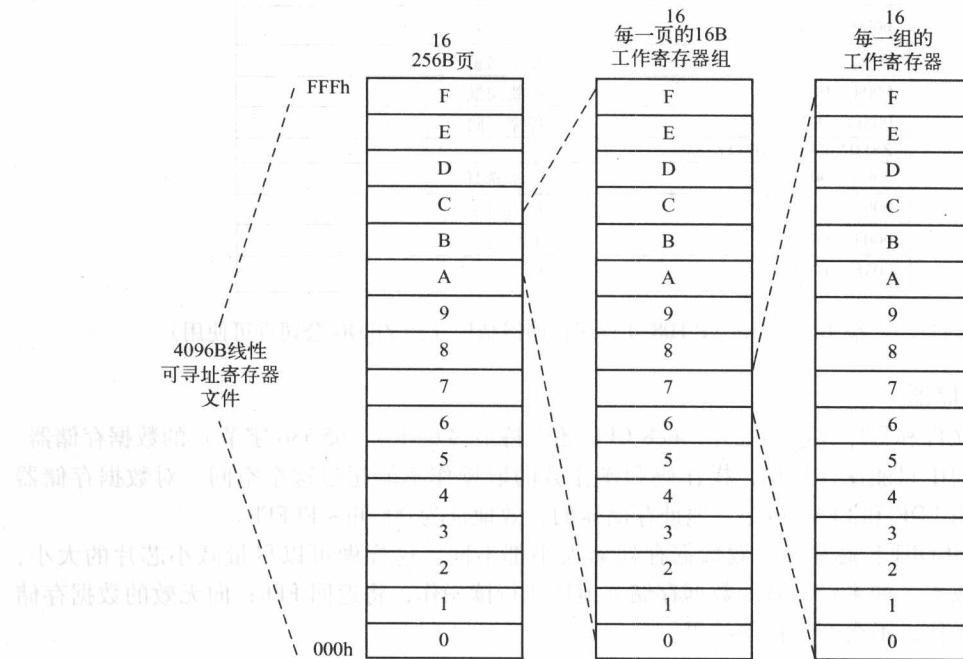


图 11-34 Z8 ENCORE! XP F0830 寄存器文件组成 (经 ZiLOG 公司许可使用)

4 KB 寄存器文件地址空间中的高 256 字节保留，用于 eZ8 CPU 的控制，包括片上外设和 I/O 端口。这些寄存器的地址空间为 F00H ~ FFFFH。当从保留的寄存器文件地址中读取数据时，会返回未定义的数据。不推荐向保留的寄存器文件地址中写数据，这样会产生意外的结果。

片上 RAM 总是从寄存器文件地址空间的 000H 地址开始。当从有效的 RAM 地址空间之外的寄存器文件地址（不在控制寄存器地址空间中）中读取数据时，会返回未定义的数据，向这些寄存器文件地址中写数据时不会产生任何结果。

## 2. 程序存储器

eZ8 CPU 支持多达 64 KB 的程序存储器地址空间。根据不同的芯片，Z8 ENCORE! XP F0830 系列芯片可以包含最多 12 KB 的片上 Flash 存储器，这些地址位于程序存储器地址空间中。当从有效的 Flash 存储器地址空间之外的程序存储器地址中读取数据时，会返回 FFH。当向这些无效的程序存储器地址中写数据时，不会有任何结果。图 11-35 所示的表格描述了程序存储器的结构。

程序存储器地址 (Hex) 功能	
Z8F0830 和 Z8F0831 产品	
0000~0001	Flash 选择位
0002~0003	复位向量
0004~003D	中断向量
003E~1FFF	程序存储器
Z8F0430 和 Z8F0431 产品	
0000~0001	Flash 选择位
0002~0003	复位向量
0004~003D	中断向量
003E~0FFF	程序存储器
Z8F0130 和 Z8F0131 产品	
0000~0001	Flash 选择位
0002~0003	复位向量
0004~003D	中断向量
003E~03FF	程序存储器
Z8F0230 和 Z8F0231 产品	
0000~0001	Flash 选择位
0002~0003	复位向量
0004~003D	中断向量
003E~07FF	程序存储器

图 11-35 Z8 ENCORE! XP F0830 程序存储器结构 (经 ZiLOG 公司许可使用)

## 3. 数据存储器

除寄存器文件和程序存储器之外，eZ8 CPU 还支持最多 64 KB (65 536 字节) 的数据存储器。数据存储器空间中只能保存数据，操作码和操作数的取操作不能超过这个空间。对数据存储器的访问可以使用 LDE 和 LEDI 指令。地址存储器的有效地址为 0000h ~ FFFFh。

每个 Z8 ENCORE! 芯片的有效数据存储器大小都不同。这样做可以尽量减小芯片的大小，并降低产品的成本。如果对无效的数据存储器地址执行读操作，将返回 FFH；向无效的数据存储器地址执行写操作，不会产生任何结果。

### 11.2.4 外设概述

图 11-36 给出了 Z8 ENCORE! XP F0830 系列产品的基本功能框图。根据所针对的应用，该系列的不同产品所包含的外设模块组合也不同。引脚较少的产品所包含的功能相对较少。这样使得设计工程师能够在功能和产品的成本之间进行平衡。

Z8 ENCORE! XP F0830 系列控制器可以将多达 17 或 25 个端口引脚（端口 A-D）配置为通用输入输出 (general-purpose input/output, GPIO)。可用的 GPIO 引脚数量根据封装大小的不同而不同，如图 11-37a 和图 11-37b 所示。其每一个引脚都是单独可编程的。

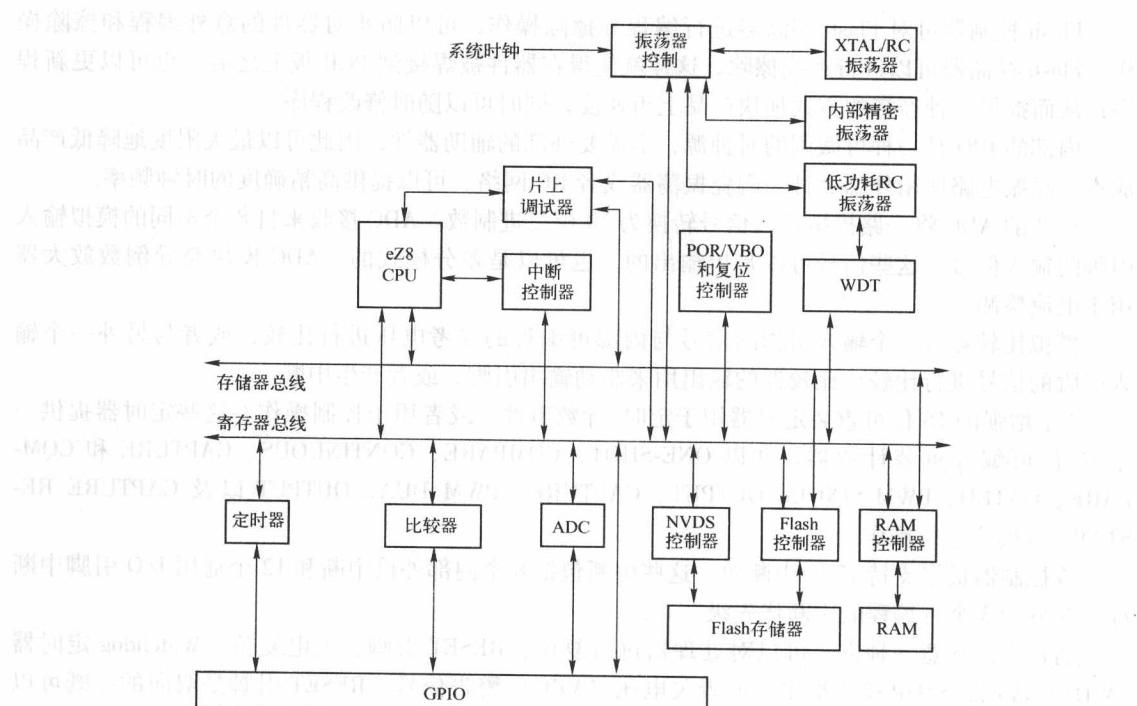


图 11-36 Z8 ENCORE! XP F0830 系列基本功能框图（经 ZiLOG 公司许可使用）

PB1/ANA1	1	20	—PB0/ANA0
PB2/ANA2	2	19	—PC3/COUT/LED
PB3/CLKIN/ANA3	3	18	—PC2/ANA6/LED/VREF
VDD	4	17	—PC1/ANA5/CINN/LED
PA0/T0IN/T0OUT/XIN	5	16	—PC0/ANA4/CINP/LED
PA1/T0OUT/XOUT	6	15	—DBG
VSS	7	14	—RESET/PD0
PA2	8	13	—PA7/T1OUT
PA3	9	12	—PA6/T1IN/TIOUT
PA4	10	11	—PA5

a) Z8 ENCORE! 20引脚的SOIC、SSOP或PDIP封装

PB2/ANA2	1	28	—PB1/ANA1
PB4/ANA7	2	27	—PB0/ANA0
PB5/VREF	3	26	—PC3/COUT/LED
PB3/CLKIN/ANA3	4	25	—PC2/ANA6/LED
(PB6)AVDD	5	24	—PC1/ANA5/CINN/LED
VDD	6	23	—PC0/ANA4/CINP/LED
PA0/T0IN/T0OUT/XIN	7	22	—DBG
PA1/T0OUT/XOUT	8	21	—RESET/PD0
VSS	9	20	—PC7/LED
(PB7)AVSS	10	19	—PC6/LED
PA2	11	18	—PA7/T1OUT
PA3	12	17	—PC5/LED
PA4	13	16	—PC4/LED
PA5	14	15	—PA6/T1IN/TIOUT

b) Z8 ENCORE! 28引脚的SOIC、SSOP或PDIP封装

图 11-37 （经 ZiLOG 公司许可使用）

Flash 控制器可对 Flash 存储器进行编程和擦除操作，可以防止对器件的意外编程和擦除操作。Flash 存储器可以进行在线擦除，这样就使得在器件被焊接到 PCB 板上之后，也可以更新程序；从而提供一种经济的方式加快产品上市速度，同时可以随时修改程序。

内部的 IPO 是一种可微调的时钟源，不需要外部的辅助器件，因此可以最大限度地降低产品成本。晶振电路使用外部金针、陶瓷振荡器或者 RC 网络，可以提供高精确度的时钟频率。

可选的 ADC 将一路模拟输入信号转换为 10 位二进制数。ADC 接收来自 8 个不同的模拟输入引脚的输入信号，这些信号可以是单输出的，也可以是差分模式的。ADC 模块跨导倒数放大器用于电流检测。

模拟比较器将一个输入引脚的信号与内部可编程的参考电压进行比较，或者与另外一个输入引脚的信号进行比较。比较器的输出用来驱动输出引脚，或者产生中断。

2 个增强的 16 位可重装定时器用于定时/计数事件，或者用于控制操作。这些定时器提供一个 16 位可编程重装计数器，并以 ONE-SHOT、COMPARE、CONTINUOUS、CAPTURE 和 COMPARE、GATED、PWM SINGLE OUTPUT、CAPTURE、PWM DUAL OUTPUT 以及 CAPTURE RE-START 等模式进行操作。

该控制器最多支持 17 个中断源。这些中断包括 8 个内部外设中断和 12 个通用 I/O 引脚中断源，共分为 3 个可编程的中断优先级。

通过下列任意一种方式可以对处理器进行复位：RESET 引脚、上电复位、Watchdog 定时器（WDT）溢出，STOP 模式推出，或者欠电压（VBO）警告信号。RESET 引脚是双向的，既可以作为复位源信号，也可以作为复位的指示信号。

处理器还支持集成的片上调试器（OCD）。OCD 具有一组丰富的调试功能，例如读和写寄存器、对 Flash 存储器进行编程、设置断点以及执行代码。OCD 仅使用一个引脚与外部主机进行通信。

### 11.2.5 复位控制器和停止模式恢复

复位控制器控制着复位和停止模式恢复操作，并且在低电压条件下给出指示。典型的情况下，如下事件会导致复位：

- 上电复位（POR）；
- 欠电压（VBO）；
- Watchdog 定时器（WDT）定时时间到（当由 WDT\_RES flash 选项位配置为初始化复位时）；
- 外部 RESET 引脚触发（当 GPIO 寄存器启用其他复位功能时）；
- 片上调试器（OCD）初始化复位（OCDCTL[0] 设置为 1）。

当控制器在 STOP 模式时，以下两种情况都会产生停止模式恢复：WDT 定时时间到，或者 GPIO 端口输入引脚转换，该输入引脚连接的是某个被设置为使能的停止模式恢复源。当供电电压下降到低于最小安全值之后，控制器上的 VBO 电路将产生 VBO 复位。

### 11.2.6 低功耗模式

在控制器设计中还引入了重要的节能特性。最节能的模式为 STOP 模式，次之为 HALT 模式，再次之是禁止某些外设模块，使其工作在 NORMAL 模式下。

执行 eZ8 CPU 的 STOP 指令后，处理器进入 STOP 模式。在 STOP 模式下，其操作特性如下：

- 主晶振和内部的精密振荡器被停止；XIN 和 XOUT（如果先前已经被启用）被禁止，PA0/PA1 引脚恢复到由 GPIO 寄存器所编程的状态；

- 系统时钟停止；
- eZ8 CPU 停止；
- 程序计数器（Program counter, PC）停止增加；
- 如果由晶振控制寄存器设置为使能，Watchdog 定时器（WDT）内部 RC 晶振继续工作；
- 如果被设置为使能，WDT 逻辑继续运行；
- 如果被相关的 flash 选项位设置为可以运行在 STOP 模式下，欠电压（VBO）保护电路继续运行；
- 其他片上的外设功能都被禁止。

为了使 STOP 模式下的电流最小，如果上拉寄存器位被设置为使能，被配置为数字输入的 GPIO 引脚必须驱动为 VDD；或者如果上拉寄存器为被设置为禁止，这些引脚必须连接到电源线上（VDD 或者 GND）。执行停止模式复位操作可以使处理器退出 STOP 模式。

当执行了 eZ8 CPU 的 HALT 指令后，处理器进入 HALT 模式。在 HALT 模式下，其操作特性如下：

- 主晶振继续运行；
- 系统时钟继续运行；
- eZ8 CPU 停止；
- 程序计数器（PC）停止增加；
- WDT 的内部 RC 晶振继续工作；
- 如果被设置为使能，WDT 逻辑继续运行；
- 其他片上的外设功能都继续运行；

执行下列操作之一都可以使 eZ8 CPU 退出 HALT 模式：

- 中断；
- WDT 定时时间到（中断或复位）；
- 上电复位（POR）；
- VBO 复位；
- 外部 RESET 引脚输入。

为了使 HALT 模式下的电流最小，所有被配置为输入的 GPIO 引脚必须驱动到某个电源（ $V_{DD}$  或  $V_{SS}$ ）。

### 11.2.7 通用输入/输出

处理器最多可以将 25 个端口引脚（端口 A-D）用作通用输入/输出（GPIO）端口。每个端口都具有控制寄存器和数据寄存器。GPIO 控制寄存器可以设置数据的方向、漏端开路、输出驱动电流、可编程上拉、停止模式恢复以及可选的引脚功能，每一个端口引脚都是单独可编程的。此外，端口 C 引脚还可以直接驱动 LED，驱动力是可编程的。

#### 1. GPIO 体系结构

图 11-38 给出了 GPIO 端口引脚的一个简单框图。图中并没有关于多种功能以及端口电流驱动力可变方面的解释。

#### 2. GPIO 的可选功能

大部分 GPIO 端口引脚都用作 GPIO 功能，用于访问片上的外设，例如定时器和串行通信设备。端口 A-D 可选功能子寄存器将这些引脚配置为 GPIO 或者可选功能操作。若一个引脚被配置为可选功能，端口引脚方向（输入/输出）将不再根据端口 A-D 数据方向寄存器来确定，而是转

换为对该端口指派的可选功能。

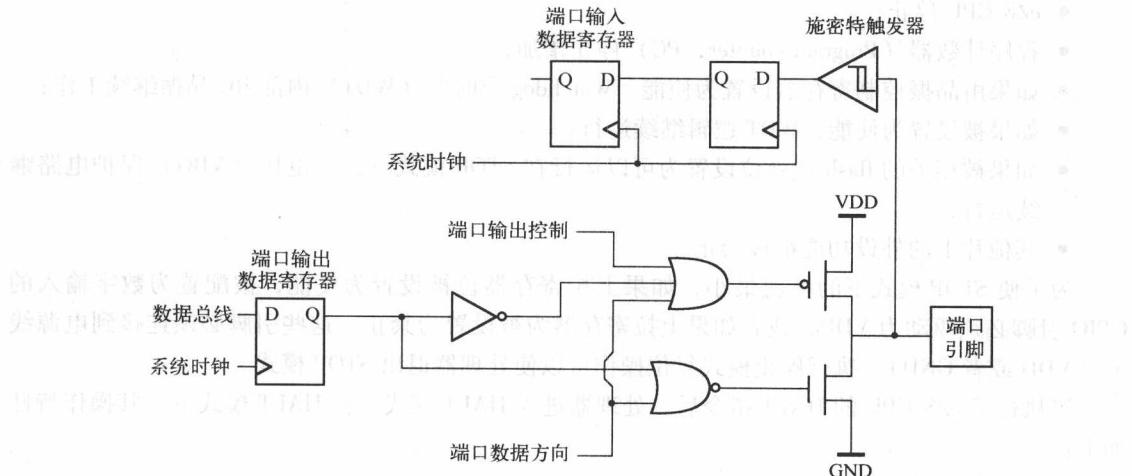


图 11-38 GPIO 框图（经 ZiLOG 公司许可使用）

晶振功能不受 GPIO 模块的控制。当晶振在振荡器控制模块中被设置为使能时，PA0 和 PA1 的 GPIO 功能将被忽略。在这种情况下，这些端口作为晶振的输入和输出。

PA0 和 PA6 具有两种不同的定时器功能，定时器输入和互补定时器输出。这两种功能都需要相同的 GPIO 配置。这两种功能要根据定时器模式进行选择。

### 3. GPIO 中断

许多 GPIO 端口引脚都用作中断源。一些端口引脚被配置为在引脚输入信号的上升沿或者下降沿产生中断请求，其他端口引脚中断源在信号的两个沿（上升沿和下降沿）都产生中断。

#### 11.2.8 中断控制器

中断控制器区分来自片上外设和 GPIO 端口引脚的中断优先级。中断控制器的体系结构如图 11-39 所示，主要特性如下：

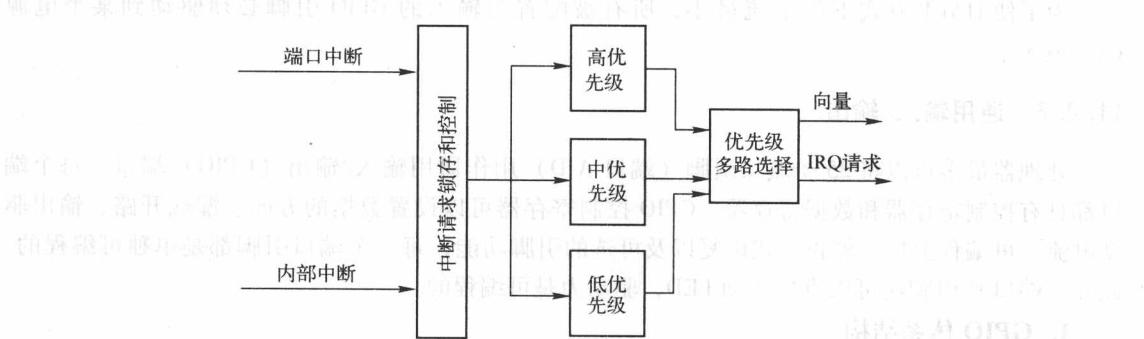


图 11-39 Z8 ENCORE! XP F0830 中断控制器框图（经 ZiLOG 公司许可使用）

- 17 个中断源使用 16 个不同的中断向量：
  - 12 个 GPIO 端口引脚中断源；
  - 5 个片上外设中断源。
- 灵活的 GPIO 中断：

- 8个可选的上升沿和下降沿GPIO中断；
- 4个双沿中断；
- 3个可单独编程的中断优先级。
- Watchdog定时器(WDT)可以被配置为中断源。

中断请求(IRQ)允许外设按顺序挂起CPU操作，并迫使CPU开始执行中断服务子程序(ISR)。通常ISR的功能包括在CPU和中断外设之间交换数据、状态信息或者控制信息。当服务子程序执行完后，CPU返回被打断的地方继续执行。

eZ8 CPU既支持矢量中断处理，也支持查询中断处理。对于查询中断，中断控制器对操作没有任何影响。中断向量被保存在偶数程序存储器地址的最高字节(MSB)，以及之后的奇数程序存储器地址的最低字节 LSB)中。一些端口中断在20引脚和28引脚的封装中并不提供。在不包含ADC功能的处理器中，不支持ADC中断。

### 1. 主设备中断使能

中断控制寄存器中的主设备中断使能(interrupt enable, IRQE)位可以全局启用或禁止中断。通过执行以下操作之一可以全局地使能中断：

- 执行使能中断(Enable Interrupt, EI)指令；
- 执行中断返回(Return from Interrupt, IRET)指令；
- 向中断控制寄存器的IRQE位写入1。

通过执行以下操作之一可以全局地禁止中断：

- 执行禁止中断(Disable Interrupt, DI)指令；
- eZ8 CPU应答来自中断控制器的中断服务请求；
- 向中断控制寄存器的IRQE位写入0；
- 复位；
- 执行陷阱指令；
- 非法的指令陷阱；
- 主振荡器陷阱；
- Watchdog振荡器陷阱。

### 2. 中断向量和优先级

中断控制器支持三个中断优先级。第三级具有最高的优先权，第二级的优先权次之，第一级的优先级最低。如果所有的中断都具有相同的中断优先级(例如，所有的中断优先级为二级)，在中断优先权从最高到最低进行分配。第三级中断总是分配为比第二级中断高的优先权，第二级中断总是分配为比第一级中断高的优先权。在每个中断优先级内部(第一级、第二级或第三级)，优先权根据相应优先级的说明进行分配。复位、Watchdog定时器中断(如果设置为使能)、主振荡器陷阱、Watchdog振荡器陷阱和非法指令陷阱总是具有最高(第三级)优先权。

中断源只在一个系统时钟周期(一个脉冲)中发出它们的中断请求。当中断请求得到了eZ8 CPU的响应后，中断请求寄存器的相应位被清除，直到下一次中断发生。向中断控制寄存器的相应位写入0，可以清除中断请求。

程序代码可以直接产生中断。向中断控制寄存器的相应位写入1，可以触发一个中断(假设中断被设置为使能)。当中断请求得到了eZ8 CPU的响应后，中断请求寄存器的相应位自动被清0。

### 11.2.9 定时器

Z8 ENCORE! XP F0830处理器中最多可以有2个16位的可重装定时器，可以用于定时、事

件计数或者产生脉宽调制（PWM）信号。这两个定时器如图 11-40 所示，具有如下功能：

- 16 位的重装计数器；
- 可编程预分频器，预分频值的范围为 1 ~ 28；
- PWM 输出产生；
- 采样和比较功能；
- 外部输入引脚作为定时器输入、时钟门控或捕捉时钟；
- 外部输入引脚信号的最高频率被限定在系统时钟频率的 1/4；
- 定时器输出引脚；
- 定时器中断。

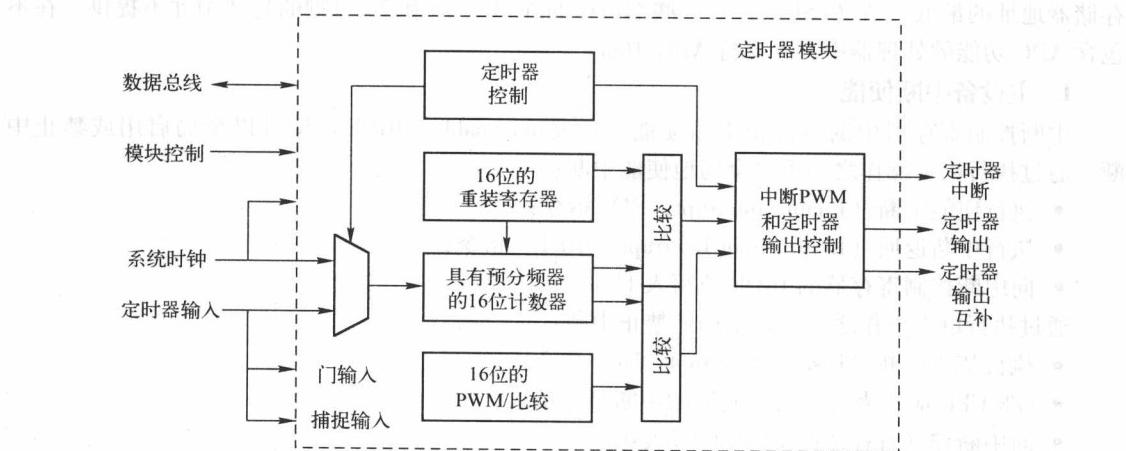


图 11-40 Z8 ENCORE! XP F0830 定时器框图（经 ZILOG 公司许可使用）

在 PWM SINGLE/DUAL 模式下，当选项位 OMPB 被设置为 0 时，比较器输出可以禁止定时器。

两个定时器都是 16 位的递增计数器。将 0001H 装载到定时重装高字节和低字节寄存器，并设置预分频器值为 1，可以设置最小的定时延迟。将 0000H 装载到定时重装高字节和低字节寄存器，并设置预分频器值为 128，可以设置最大的定时延迟。如果定时器到达 FFFFH，将恢复到 0000H，并继续计数。

### 1. ONE-SHOT 模式

在 ONE-SHOT 模式中，定时器计数到 16 位的重装值，重装值保存在定时器重装高字节和低字节寄存器中。定时器的输入是系统时钟。当计数到达重装值后，定时器产生一个中断，定时器高和低字节寄存器中的计数值被恢复为 0001H。定时器自动被禁止，并停止计数。

如果定时器输出可选功能被设置为使能，在定时器重装时，每一个系统时钟周期（从低到高或从高到低）定时器输出引脚都会改变状态。

$$\text{ONE-SHOT 模式定时时间到周期 (s)} = \frac{(\text{重装值} - \text{起始值}) \times \text{预分频}}{\text{系统时钟频率 (Hz)}}$$

### 2. CONTINUOUS (连续) 模式

在 CONTINUOUS 模式中，定时器计数到 16 位的重装值，重装值保存在定时器重装高字节和低字节寄存器中。定时器的输入是系统时钟。当计数到达重装值后，定时器产生一个中断，定时器高和低字节寄存器中的计数值被恢复为 0001H，并继续计数。如果定时器输出其他功能被设置为使能，在定时器重装时，每一个系统时钟周期（从低到高或从高到低）定时器输出引脚都会

改变状态。

$$\text{CONTINUOUS 模式定时时间到周期 (s)} = \frac{\text{重装值} - \text{预分频}}{\text{系统时钟频率 (Hz)}}$$

### 3. COMPARATOR COUNTER (比较器计数器) 模式

在 COMPARATOR COUNTER 模式中，定时器计数模拟比较器输出输入的转换次数。定时器控制寄存器的 TPOL 位选择是在比较器输出信号的上升沿计数，还是在下降沿计数。在 COMPARATOR COUNTER 模式中，预分频器被禁止。比较器输出信号的频率必须小于系统时钟频率的 1/4。定时器计数到达保存在定时器重装高字节和低字节寄存器中的重装值后，定时器产生一个中断。定时器高和低字节寄存器中的计数值被恢复为 0001H，并继续计数。如果定时器输出其他功能被设置为使能，在定时器重装时，每一个系统时钟周期（从低到高或从高到低）定时器输出引脚都会改变状态。

在 COMPARATOR COUNTER 模式中，从定时器启动后的比较器输出转换的次数由如下公式给出：

$$\text{比较器输出转换} = \text{当前计数值} - \text{起始值}$$

### 4. PWM 单数差模式

在 PWM 单数差模式中，定时器通过 GPIO 端口引脚输出 PWM 输出信号。定时器输入是系统时钟。定时器首先计数到保存在定时器 PWM 高字节和低字节寄存器中的 16 位 PWM 匹配值。当定时器计数值与 PWM 值相匹配时，定时器的输出发生翻转；定时器继续计数，直到到达保存在定时器 PWM 高字节和低字节寄存器中的 16 位 PWM 匹配值。当到达了重装值后，定时器产生一个中断，定时器高和低字节寄存器中的计数值被恢复为 0001H，并继续计数。

如果定时器控制寄存器中的 TPOL 位被设置为 1，当定时器值与 PWM 值相匹配时，定时器输出信号从高（1）转换为低（0）；当定时器到达重装值后，定时器输出信号恢复为高（1），重装值被复位为 0001H。如果定时器控制寄存器中的 TPOL 位设置为 0，当定时器值与 PWM 值相匹配时，定时器输出信号从低（0）转换为高（1）；当定时器到达重装值后，定时器输出信号恢复为低（0），重装值被复位为 0001H。

$$\text{PWM 周期} = \frac{\text{重装值} - \text{预分频}}{\text{系统时钟频率 (Hz)}}$$

### 5. PWM 双输出模式

在 PWM 双输出模式中，定时器通过两个 GPIO 端口引脚输出一对 PWM 输出信号（基本 PWM 信号和它的互补信号）。定时器的输入是系统时钟。定时器首先计数到保存在定时器 PWM 高字节和低字节寄存器中的 16 位 PWM 匹配值。当定时器计数值与 PWM 值相匹配时，定时器的输出发生翻转；定时器继续计数，直到到达保存在定时器 PWM 高字节和低字节寄存器中的重装值。当到达了重装值后，定时器产生一个中断，定时器高和低字节寄存器中的计数值被恢复为 0001H，并继续计数。

定时器还产生另外一个 PWM 输出信号，即定时器输出互补信号。定时器输出互补信号是定时器输出的 PWM 信号的互补。可编程死区延迟，可以配置为时间延迟（0 ~ 128 个系统时钟周期）。PWM 输出在这两个引脚上从低跳转到高（从停止到活跃），这样可以确保一个 PWM 输出的消除到互补信号的发送之间有一个时间间隙。

PWM 周期由如下公式给出：

$$\text{PWM 周期} = \frac{\text{重装值} - \text{预分频}}{\text{系统时钟频率 (Hz)}}$$

## 6. 捕捉模式

在捕捉模式中，相应的外部定时器输入转换发生后，当前定时器计数值被记录下来，捕捉计数值被写入到 PWM 高字节和定时器 PWM 低字节寄存器中。定时器的输入是系统时钟，定时器控制寄存器中的 TPOL 位确定捕捉是发生在定时器输入信号的上升沿还是下降沿。捕捉事件发生后，会产生一个中断，定时器继续计数。对 TxCTL1 寄存器中的 INPCAP 位进行设置，表示定时器中断是由输入捕捉事件触发的。

定时器首先计数到保存在定时器重装高字节和低字节寄存器中的 16 位重装值。当定时器计数值与重装值相匹配时，定时器会产生一个中断，并继续计数。TxCTL1 寄存器中的 INPCAP 位被清除，表明定时器中断并不是由输入捕捉事件触发的。

## 7. 捕捉重启模式

在捕捉重启模式中，相应的外部定时器输入翻转发生后，当前定时器计数值被记录下来，捕捉计数值被写入到定时器 PWM 高字节和低字节寄存器中。定时器的输入是系统时钟。定时器控制寄存器中的 TPOL 位确定捕捉是发生在定时器输入信号的上升沿还是下降沿。捕捉事件发生后，会产生一个中断，定时器高字节和低字节寄存器中的计数值被复位为 0001H，定时器继续计数。对 TxCTL1 寄存器中的 INPCAP 位进行设置，表示定时器中断是由输入捕捉事件触发的。

如果没有发生捕捉事件，定时器计数到保存在定时器重装高字节和定时器重装低字节寄存器中的 16 位比较值。当计数到达重装值时，定时器产生一个中断，定时器高字节和低字节寄存器中的计数值被复位为 0001H，定时器继续计数。TxCTL1 寄存器中的 INPCAP 位被清除，表示定时器中断不是由输入捕捉事件触发的。

在捕捉模式中，从定时器启动到捕捉事件发生时所经历的时间由如下公式计算：

$$\text{捕捉时间 (s)} = \frac{(\text{捕获值} - \text{起始值}) \times \text{预分频}}{\text{系统时钟频率 (Hz)}}$$

## 8. 比较模式

在比较模式中，定时器计数到保存在定时器重装高字节和定时器重装低字节寄存器中的 16 位最高比较值。定时器的输入是系统时钟。当计数到达比较值时，定时器会产生一个中断，并继续计数（定时器值不复位到 0001H）。如果定时器输出其他功能被设置为使能，在比较时，定时器输出引脚状态会发生变化（从低到高或从高到低）。如果定时器计数值到达了 FFFFH，定时器计数值将被复位到 0000H，并继续计数。

在比较模式中，系统时钟作为定时器的输入，比较时间由如下公式进行计算：

$$\text{比较模式时间 (s)} = \frac{(\text{比较值} - \text{起始值}) \times \text{预分频}}{\text{系统时钟频率 (Hz)}}$$

## 9. 门控模式

在门控模式中，只有在定时器输入信号处于有效状态（需要进行设置）时，定时器才会计数，有效状态是由定时器控制寄存器中的 TPOL 位设置的。当定时器输入信号发出时，计数开始；当定时器输入信号停止发送或者发生定时器重装时，就会产生定时器中断。要确定定时器输入信号消除是否产生了中断，可以读取相应的 GPIO 输入值，并将其与保存在 TPOL 位中的值进行比较。定时器首先计数到保存在定时器重装高字节和低字节寄存器中的 16 位重装值。定时器的输入是系统时钟。当计数到达比较值时，定时器会产生一个中断，定时器高字节和低字节寄存器中的计数值被复位到 0001H，并继续计数（假设定时器输入信号保持发送）。

如果定时器输出其他功能被设置为使能，在定时器复位时，定时器输出引脚状态会发生变化（从低到高或从高到低）。

## 10. 捕捉/比较模式

在捕捉/比较模式中，当外部定时器输入信号第一次发生翻转时，定时器开始计数。有效的翻转（上升沿或下降沿）由定时器控制寄存器中的 TPOL 位进行设置。定时器的输入是系统时钟。定时器输入信号的每一个有效翻转序列（第一次翻转之后）都会采样当前的计数值。采样的计数值被写入到定时器 PWM 高字节和低字节寄存器中。当采样时间发生时，会产生一个中断，定时器高字节和低字节寄存器中的计数值会复位为 0001H，并继续计数。对 TxCTL1 寄存器中的 INPCAP 位进行设置，表示定时器中断是由输入捕捉事件触发的。

如果没有发生捕捉事件，定时器首先计数到保存在定时器重装高字节和低字节寄存器中的 16 位比较值。当计数到达比较值时，定时器产生一个中断，定时器高字节和低字节寄存器中的计数值会复位为 0001H，并继续计数。TxCTL1 寄存器中的 INPCAP 位被清除置，表示定时器中断不是由输入捕捉事件触发的。

在捕捉/比较模式中，从定时器启动到采样事件发生时所经历的时间由如下公式计算：

$$\text{捕捉时间 (s)} = \frac{(\text{捕获值} - \text{起始值}) \times \text{预分频}}{\text{系统时钟频率 (Hz)}}$$

### 11.2.10 Watchdog 定时器

Watchdog 定时器（WDT）对不可靠的软件、电源故障和其他系统级问题进行预防和保护，这些情况都会导致硬件进入错误的工作状态。当发生这些情况时，程序计数器可以被复位，程序重新正常执行。WDT 具有如下特点：

- 片上 RC 振荡器；
- 可选择的定时时间到响应：复位或中断；
- 24 位可编程定时时间值。

WDT 是一种可重触发的 ONE SHOT 定时器，WDT 到达计数值后，会产生复位或中断。WDT 有专用的片上 RC 振荡器作为它的时钟源，只有 ON 和 OFF 2 种操作模式。

一旦被设置为使能，WDT 总是计数，并且必须不断刷新，以避免产生定时时间到。执行 WDT 指令或者设置 WDT\_AO Flash 选择位，可以将 WDT 设置为使能。即使并没有执行 WDT 指令，WDT\_AO 位迫使 WDT 在复位后马上运行。WDT 是一个 24 位的可重装递减计数器，用 eZ8 CPU 寄存器空间中的 3 个 8 位寄存器设置重装值。最小的 WDT 定时时间由如下公式描述：

$$\text{WDT 定时时间周期 (s)} = \frac{\text{WDT 重装值}}{10}$$

WDT 重装值是由 {WDTU[7:0], WDTH[7:0], WDTL[7:0]} 所给出的 24 位值的十进制数，典型的 Watchdog 定时器 RC 振荡器频率为 10 kHz。当计数到达 000002H 后，不能对 WDT 进行刷新。

### 11.2.11 模数转换器

在处理器的外设功能模块中有一个 8 通道连续逼近寄存器（successive approximation register，SAR）模数转换器（ADC）。ADC 可将一路模拟输入信号转换为一个 10 位的二进制数据。SAR ADC 的特点有：

- 8 个模拟输入信号源，复用通用 I/O 端口；
- 快速转换时间，小于 11.9 μs；
- 可编程定时控制；
- 转换完成后产生中断；
- 内部电压参考产生器；

- 可以选择外部参考电压。

使用外部参考电压配置 ADC 时，在 28 引脚封装中，PB5 被用作参考电压；在 20 引脚封装中，PC2 (ANA6) 被用作参考电压。

ADC 体系结构如图 11-41 所示，包含一个 8 输入多路器、采样和保持放大器和一个 10 位的 SAR ADC。ADC 将所选通道的信号进行数字化，并将数字化数据保存在 ADC 数据寄存器中。在电噪声比较大的环境中，必须在输入引脚上增加外部 RC 滤波器，以减小高频噪声。

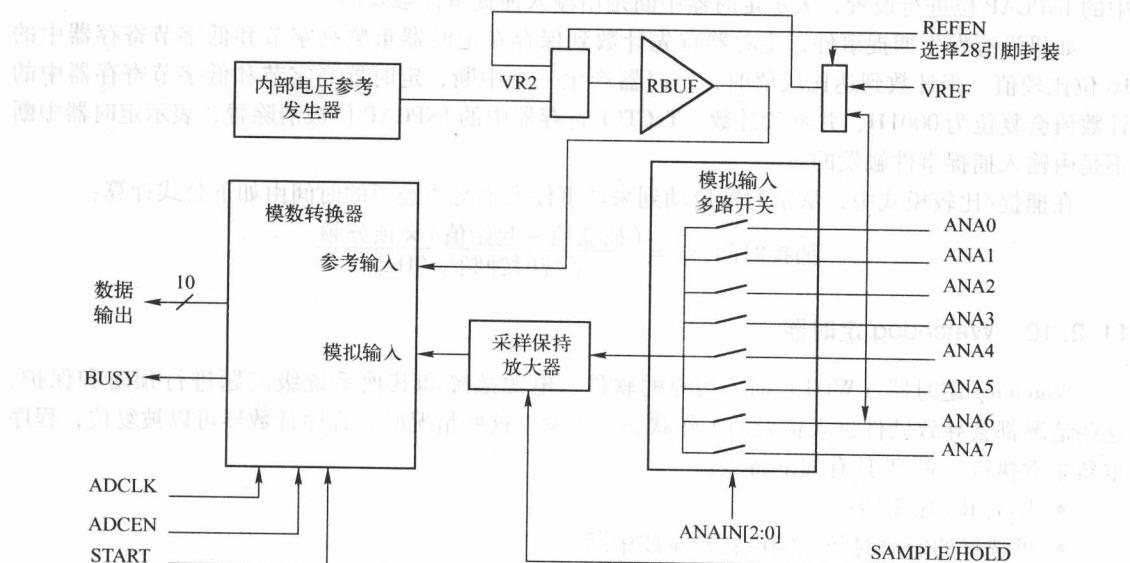


图 11-41 Z8 ENCORE! XP F0830 模数转换器框图（经 ZILOG 公司许可使用）

### 1. ADC 操作

ADC 将模拟输入 ANAX 转换为一个 10 位的数字表示。计算数字值的公式为：

$$\text{ADCOutput} = 1024 \times (\text{ANA}_X \div V_{\text{REF}})$$

假设 0 增益和偏移量错误，任何超出 ADC 输入限制的电压，即  $A_{\text{vss}}$  和参考电压，将分别返回全 0 或全 1。软件对通过写 ADC 控制寄存器的起始位，可以初始化一次新的转换。初始化操作将阻止任何当前执行的转换过程，并开始一次新的转换。为了避免打断已经开始的转换过程，可以读取 START 位，以检查 ADC 操作的状态（忙或闲）。

### 2. ADC 定时

ADC 转换的过程由如下 3 步组成：

第一步：输入采样（可编程的，最小  $1.0 \mu\text{s}$ ）；

第二步：采样保持放大器 settling（可编程的，最小  $0.5 \mu\text{s}$ ）；

第三步：转换需要 13ADCLK 周期，图 11-42a 和图 11-42b 给出了一次 ADC 转换的时序。

### 11.2.12 比较器

Z8 ENCORE! XP F0830 系列处理器中有一个通用比较器，可以用来比较两个模拟输入信号。GPIO (CINP) 引脚提供正的比较器输入，负的输入 (CINN) 可以来自外部 GPIO 引脚或内部参考。比较器的输出可以作为一个中断源，或者可以使用 GPIO 多路器输出到外部引脚上。

比较器的特点如下：

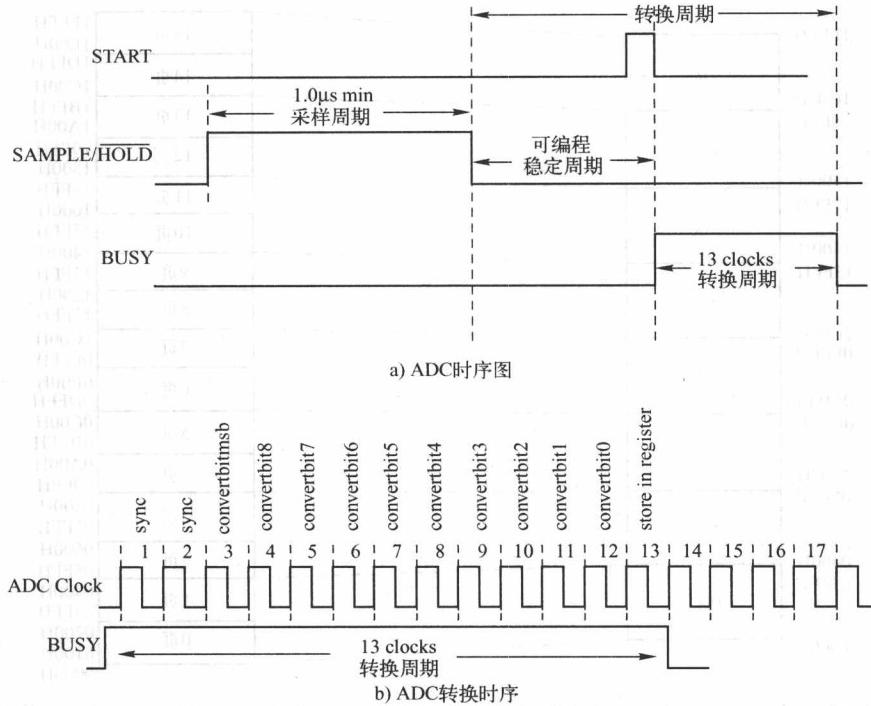


图 11-42 (经 ZiLOG 公司许可使用)

- 正的输入连接到 GPIO 引脚；
- 负的输入可以连接到外 GPIO 引脚，也可以是可编程的内部参考；
- 输出可以作为中断源，或者可以输出到外部引脚。

比较器的一个输入连接到内部参考，这是一个用户可选的参考，并且可以 200 mV 的精度进行编程。为了降低电流，比较器处于节点模式。由于比较器的传播延迟，在中断禁止的情况下，不推荐启用比较器，并等待比较器输出。这种延迟可以在启用比较器之后避免虚假中断的发生。

### 11.2.13 Flash 存储器

Z8 ENCORE! XP F0830 系列处理器的非易失性 Flash 存储器的容量可以是 1KB、2KB、4KB、8KB（具有 NVDS）或者 12KB（没有 NVDS），这些存储器都可以进行读/写/擦除操作。通过用户代码或片上编程器，可以现场对 Flash 存储器进行编程和擦除。

Flash 存储器阵列按页组织，每页有 512 个字节。512 字节的页是可以被擦除的最小 Flash 块，每一页被分为 8 行，每行 84 字节。为了便于程序/数据保护，Flash 存储器还可以按扇区组织，每一个扇区可以映射到 1 页（对于 1KB、2KB 和 4KB 的处理器）、2 页（8KB 的处理器）或者 3 页（12KB 的处理器）。

图 11-43 给出了 8KB 处理器的 Flash 存储器组织。

可编程 Flash 选项位允许用户对器件操作的某些方面进行硬配置。特性配置数据保存在 Flash 程序存储器中，在复位过程中读取。Flash 选项位可以控制的特性如下：

- WDT 定时时间到响应选择——中断或者系统复位。
- 复位时使能 WDT。
- 阻止对程序存储器中用户代码的非法访问。
- 阻止对程序存储器中的全部或部分用户代码的意外编程或擦除。

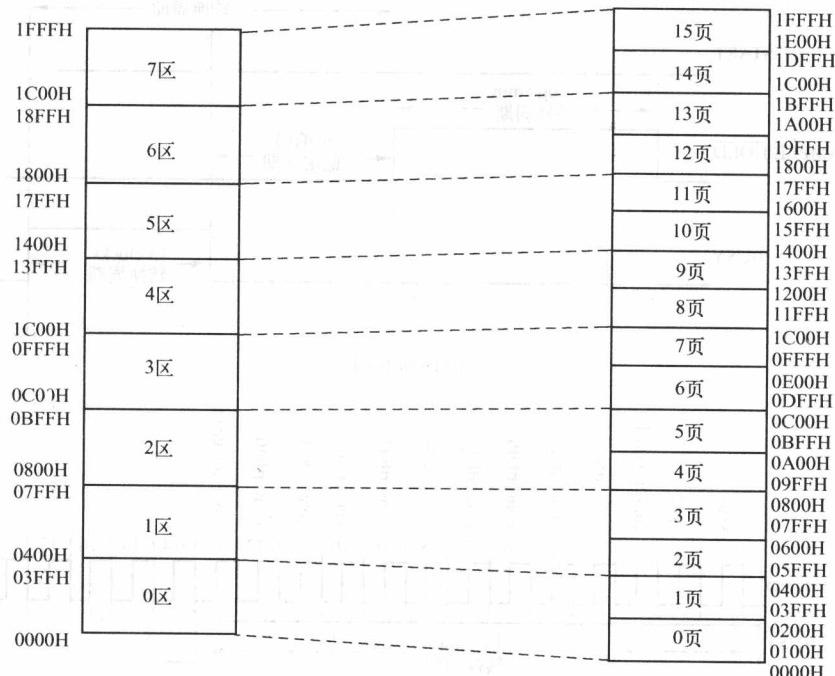


图 11-43 具有 NVDS 的 Z8 ENCORE! XP F0830 8K Flash 组织（经 ZiLOG 公司许可使用）

- 在 STOP 模式下，通常可以使能或禁止欠电压（VBO）配置，以减小 STOP 模式的功耗。
- 振荡器模式选择，可以有高、中和低功耗晶体振荡器，或者外部 RC 振荡器。
- 厂家信息，内部精度振荡器和 VBO 电压。

### 11.2.14 非易失性数据存储

处理器支持多达 64 字节的非易失性数据存储（NVDS）（Flash 12KB 模式例外）。这种存储器可以执行 100 000 以上次的写操作，可以用作常数存储器。

设计人员不能访问的程序存储器区域中可以保存专用 ZiLOG 软件，用以实现 NVDS。这些专用子程序使用 Flash 存储器来保存数据。子程序中采用一种动态寻址方案，以使 Flash 的写/擦除更持久。

在访问 NVDS 时，需要两个子程序，分别是写子程序和读子程序。这两个子程序都可以通过一条 CALL 指令调用，该指令的目标地址为程序存储器空间之外预先定义好的地址。NVDS 地址和数据都是单字节的值。为了不打乱用户代码，这些子程序在使用工作寄存器组之前，都要先进行保存，因此需要一个 16 字节的堆栈空间。在完成对这些子程序的调用操作之后，用户代码的工作寄存器组就可以恢复。

对 NVDS 的读和写操作过程中，禁止中断服务功能。在 NVDS 执行过程中所产生的中断，不能打乱工作寄存器和当前堆栈所保存的内容，否则存储器阵列将被破坏，推荐设计者在执行 NVDS 操作之前将中断禁止。NVDS 的使用需要 16 字节的可用堆栈空间。在调用 NVDS 读或写子程序之前，需要保存工作寄存器组的值。

### 11.2.15 片上调试器

处理器还有一个重要的特点，就是集成的片上调试器（OCD），如图 11-44 所示。该调试器提供了如下先进的调试功能，包括：

- 寄存器文件的读和写；
- 程序存储器和数据存储器的读和写；
- 断点和观察点的设置；
- 执行 eZ8 CPU 指令。

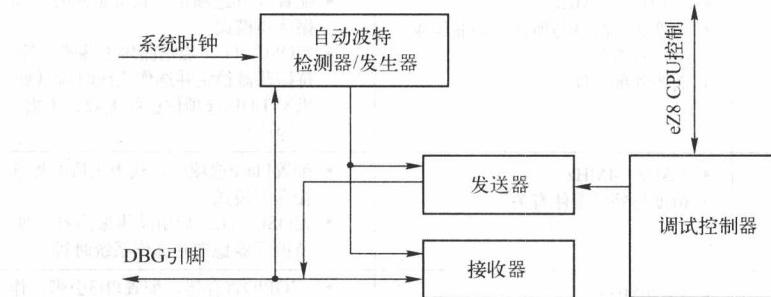


图 11-44 Z8 ENCORE! XP F0830 片上调试器框图（经 ZiLOG 公司许可使用）

OCD 共包括发送器、接收器、自动波特检测器/产生器，调试控制器 4 个主要的功能模块。

OCD 使用 DBG 引脚与外界主机进行通信。这种单引脚接口是双向的 open-drain 接口，可以发送和结束数据。数据传送是半双工的，发送和接收不能同时进行。DBG 引脚的串行数据使用 RS-232 中定义的标准异步数据格式发送，该引脚可以使用最小的外部硬件创建一个与 PC 串口的接口。

### 11.2.16 振荡器控制

处理器共支持如下 5 种时钟方案，并且都是用户可选的：

- 片上精确调整 RC 振荡器；
- 使用片内晶体或振荡器的片上振荡器；
- 使用片外 RC 网络的片上振荡器；
- 外部时钟驱动；
- 片上低精度 WDT 振荡器。

此外，还设计了时钟失效检查和恢复电路，这样，即使主振荡器发生故障，也可以继续执行。图 11-45 中的表给出了振荡器的配置和选择。

#### 1. 晶体振荡器

在芯片中提供了一个多功能晶体振荡器，可以与频率范围在 32 kHz ~ 20 MHz 的外部晶体一起使用。此外，振荡器支持振荡频率最高为 4MHz 的外部 RC 网络，或者最高频率为 8 MHz 的陶瓷振荡器。片上晶体振荡器可以为 eZ8 CPU 和大多数片上外设产生主系统时钟。

XIN 输入引脚也可以接收 CMOS 级的时钟输入信号 (32 kHz ~ 20 MHz)。如果使用外部时钟产生器，XOUT 引脚必须处于空闲状态。片内晶体振荡器还具有时钟滤波器功能，但是默认情况下，该时钟滤波器处于禁止状态，对输入时钟没有分频。也就是说在默认设置下，XIN 输入引脚上的信号频率决定了系统时钟频率。

采用用户可编程的 Flash 选项位可以选择晶体振荡器模式。系统支持的 4 种晶体振荡器模式有：

- 功耗最小的低频晶体 (32 kHz ~ 1 MHz)；
- 中等功耗的中频晶体或陶瓷振荡器 (0.5 ~ 8 MHz)；
- 最高功耗的高频晶体 (8 ~ 20 MHz)；
- 片上振荡器，配置为与外部 RC 网络一起使用 (<4 MHz)。

时钟源	特性	需要的设置
内部精度RC振荡器	<ul style="list-style-type: none"> <li>• 32.8 kHz或5.53MHz</li> <li>• <math>\pm 4\%</math>精度</li> <li>• 不需要外部器件</li> </ul>	<ul style="list-style-type: none"> <li>• 写振荡器控制寄存器 (OSCCTL)，选择频率为5.53MHz或32.8 kHz的振荡器</li> </ul>
外部晶体振荡器	<ul style="list-style-type: none"> <li>• 32 kHz~20MHz</li> <li>• 精度非常高（与所使用的晶体或振荡器有关）</li> <li>• 需要外部器件</li> </ul>	<ul style="list-style-type: none"> <li>• 配置Flash选项位，获得正确的外部振荡器模式</li> <li>• 写OSCCTL，启用晶体振荡器，等待振荡器稳定并选作系统时钟（如果XTLDIS选项位已经无效，不需要等待）</li> </ul>
外部RC振荡器	<ul style="list-style-type: none"> <li>• 32kHz~4MHz</li> <li>• 精度与外部器件有关</li> </ul>	<ul style="list-style-type: none"> <li>• 配置Flash选项位，获得正确的外部振荡器模式</li> <li>• 写OSCCTL，启用晶体振荡器，等待振荡器稳定并选作系统时钟</li> </ul>
外部时钟驱动	<ul style="list-style-type: none"> <li>• 0~20MHz</li> <li>• 精度与外部时钟源有关</li> </ul>	<ul style="list-style-type: none"> <li>• 写GPIO寄存器，配置PB3引脚，作为外部时钟使用</li> <li>• 写OSCCTL，选作外部系统时钟</li> <li>• 通过GPIO输入外部时钟信号</li> </ul>
内部Watchdog 定时器振荡器	<ul style="list-style-type: none"> <li>• 10 kHz最小</li> <li>• <math>\pm 4\%</math>精度，不需要外部器件</li> <li>• 低功耗</li> </ul>	<ul style="list-style-type: none"> <li>• 如果没有启用，先启用WDT，并等待WDT振荡器运行起来</li> <li>• 写OSCCTL，启用并选择振荡器</li> </ul>

图 11-45 Z8 ENCORE! XP F0830 振荡器配置和选择（经 ZiLOG 公司许可使用）

## 2. 内部精密振荡器

内部精度振荡器 (IPO) 的设计可以不使用外部器件，从而节省设计成本。非标准频率的振荡器可以手工调整或者厂家自动调整，可以在正常工作温度下，获得 5.53 MHz 的频率，精度为 4%，占空比为 45% ~ 55%，并为设备提供电压。IPO 启动的最大时间为 25  $\mu$ s。

IPO 的特点有：

- 片上 RC 振荡器，不需要外部器件；
- 5.53 MHz 或 32.8 kHz 的输出频率（包括 FAST 和 SLOW 两种模式）；
- 可以由用户通过 Flash 选项进行调整；
- 在不需要高时序精度的应用中，可以不使用晶体或陶瓷振荡器。

内部振荡器是一种 RC 张弛振荡器，对供电变化的敏感度很低。通过使用系数跟踪门限，可以消除电源电压的影响。振荡器的主要故障来自于芯片级部件产品的绝对变化，例如电容，在设计中所包含的 8 位调整寄存器可以补偿振荡器频率的绝对变化。一旦经过调整，振荡器的频率就会稳定，不再需要校准。调整是在生产过程中完成的，除非需要使用 5.53 MHz (FAST 模式) 或 32.8 kHz (SLOW 模式) 以外的频率，否则用户没有必要做重复性的工作。

### 11.2.17 eZ8 CPU 指令和编程

处理器的精简指令集可以广泛适用于各种应用，对各种应用的编程提供足够的灵活性支持。该指令集从早期的 Z8 处理器发展而来，并实现了 Z8 处理器指令的一个超子集。

除了 HALT 模式下的 Watchdog 定时器使能指令 (WDh，操作码为 4Fh) 之外，eZ8 CPU 可以执行所有的 Z8 汇编语言指令。用户的 Z8 汇编代码可以编译成适合于 eZ8 CPU 运行的代码。eZ8 CPU 的编译器可以从 [www.zilog.com](http://www.zilog.com) 下载。

与 Z8 CPU 指令集相比，eZ8 CPU 增加了许多新的指令，能够提高处理器的工作效率，并且

可以访问扩展的4KB寄存器文件。按照功能，指令集中的83条指令可以分为如下8种类型：

- 运算；
- 位操作；
- 块传输；
- CPU控制；
- 装载；
- 逻辑；
- 程序控制；
- 旋转和移位。

处理器共支持6种寻址模式，为程序设计提供更大的灵活性和更高的效率。这样可以使得代码更加简洁，执行速度更快。这6种寻址方式为：

- 寄存器（Register, R）；
- 寄存器间接（Indirect register, IR）；
- 索引（Indexed, X）；
- 直接（Direct, DA）；
- 相对（Relative, RA）；
- 立即数（Immediate data, IM）；
- 扩展寄存器（Extended register, ER）。

### 程序堆栈

处理器实行的是一种堆栈编程体系结构。堆栈操作在寄存器文件的通用寄存器中执行。eZ8 CPU允许用户在寄存器文件中重定位堆栈。堆栈可以位于000H~EFFH的地址中。12位的堆栈指针（stack pointer, SP）值由{SPH[3:0], SPL[7:0]}确定。SP具有12位的堆栈操作增量/减量功能，可以使SP对一页以上进行操作。

堆栈操作在寄存器文件的通用寄存器中执行。寄存器组FFEH和FFFH构成了16位的SP，用于所有堆栈操作。SP中保存着当前堆栈地址。SP必须总是指向寄存器文件的一部分，不会导致用户程序数据被覆盖。即使是对于线性程序代码来说，没有用堆栈用于调用和/或中断子程序的实现手段，在对SP进行设置时，也必须考虑到非法的指令陷阱。

在执行PUSH操作之前，要对堆栈地址进行减操作，而对堆栈指针的加操作要在POP操作之后。堆栈地址总是指向栈顶元素。堆栈是中断以及CALL和TRAP指令的返回栈，还可以将其用作数据堆栈。在执行CALL指令的过程中，程序计数器的值可以保存在堆栈中。当执行返回（execution of a Return, RET）指令时，恢复程序计数器的值。

中断和陷阱（TRAP指令或者非法的指令陷阱）都会将程序计数器和标志寄存器的值保存到堆栈中，中断返回（IRET）指令对它们进行恢复。图11-46给出了调用、中断和陷阱操作之后的堆栈内容和SP位置。

在对堆栈地址进行加或减操作时，如果结果超出了规定的地址空间，会发生上溢或者下溢。程序员必须避免这种情况的发生，否则会产生意外的结果。

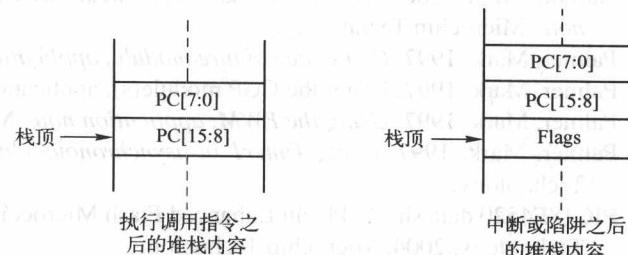


图11-46 堆栈指针寄存器（经ZiLOG公司许可使用）

## 习题

1. 解释 GPM 与 SoC 相比的优越性。
2. 列出 PIC18F4520 的主要功能块。
3. 描述 PIC18F4520 的指令流水线。
4. PIC18F4520 的配置寄存器都有哪些功能？
5. 描述 PIC18F4520 在 3 种电源模式下的区别。
6. 在 PIC18F4520 中使用 Flash 程序存储器的主要优点有哪些？
7. 在 PIC18F4520 中为什么要使用两个中断向量？
8. 三态 I/O 端口的作用是什么？
9. 在 PIC18F4520 中使用定时器的 3 个优点有哪些？
10. 画图说明使用捕捉/比较功能的一个应用。
11. 在应用中如何使用脉宽调制？
12. 为什么在 PIC18F4520 中具有多种串行 I/O 模式？
13. 设置片上 A/D 转换器有何价值？
14. 列出 PIC18F4520 的 5 种主要指令类型。
15. 列出 Zilog Z8 Encore! CPU 的关键功能模块。
16. PIC 18F4520 和 Zilog Z8 Encore! 之间在体系结构有哪些关键的区别？
17. eZ8 CPU 在什么时候会产生延迟周期？
18. 列出 eZ8 CPU 的主要寻址方式。
19. 在 eZ8 CPU 中设计三级中断方案有哪些优点？
20. 为什么在 Z8 Encore! XP F0830 中要设置三种电源管理模式？
21. 描述在 CONTINUOUS 模式下使用 Z8 Encore! XP F0830 定时器的一种应用。
22. Z8 Encore! XP F0830 中 WDT 最重要的用途是什么？
23. Z8 Encore! XP F0830 的 5 种时钟方案有哪些？

## 参考文献

- Customer presentation. 2006. Microchip Technology, Marketing Department.
- Fischer, Richard L. 2003. *Using the PICmicro MSSP module for master I<sup>2</sup>C communications, application note*. Microchip Technology.
- Garbutt, Mike. 2003. *Asynchronous communications with the PICmicro USART, application note*. Microchip Technology.
- Palmer, Mark. 1997. *Using the capture module, application note*. Microchip Technology.
- Palmer, Mark. 1997. *Using the CCP module(s), application note*. Microchip Technology.
- Palmer, Mark. 1997. *Using the PWM, application note*. Microchip Technology.
- Palmer, Mark. 1997. *Using Timer1 in asynchronous clock mode, application note*. Microchip Technology.
- PIC18F4520 data sheet, 44-Pin Enhanced Flash Microcontrollers with 10-Bit A/D and nanoWatt Technology. 2004. Microchip Technology.
- Z8 Encore! Microcontrollers eZ8 CPU Core. *User Manual UM012815-0606*. June 2006. ZiLOG, Inc.
- Z8 Encore! XP F0830 Series with Extended Peripherals. *Product brief, PB016107-0706*. July 2006. ZiLOG Inc.
- Z8 Encore! XP F0830 Series with Extended Peripherals. *Product specification, PS024707-0606*. June 2006. ZiLOG Inc.

## | 第 12 章 |

# 16 位微控制器

- 本章目标：介绍一款通用 16 位微控制器
- 学习内容：
  1. Freescale S12XD 系列嵌入式微控制器。
  2. Texas Instruments MSP430™ 系列混合信号微控制器。

## 12.0 16 位处理器概述

16 位微控制器的性能要高于 8 位微控制器。16 位的微控制器主要应用在一些需要较高处理能力的应用中，而同时价格要低于 32 位的微控制器。与 8 位微控制器相同，16 位微控制器的外设模块也相当灵活。

Freescale S12XD 系列微控制器是从早期微控制器进化而来的可以解决系统工程师在实际的设计中所遇到的挑战。S12XD 系列微控制器加深了低价、低功耗、低 EMC (electromagnetic compatibility, 电磁兼容)、高代码紧凑性和高灵活性等特性，而这些特性在汽车领域应用中是非常重要的。S12XD 系列微控制器基于增强的 HCS12 内核，其性能是 25 - MHz HCS12 微控制器的 2 ~ 5 倍，而同时又与 HCS12 微控制器保持非常高的代码兼容性。

Texas Instruments MSP430 解决方案利用了 16 位 RISC (精简指令集计算机) CPU，以获得高性能。该系列具有 5 种不同的操作模式，适用于超低功耗应用。在设计时考虑了 3 种特殊的应用，即超低功耗、速度和数据吞吐量和最小的单外设电流消耗；通过适当的中断驱动设计，芯片的供电时间可以保持最小，以节约功耗，同时具有较高的系统级性能。

## 12.1 Freescale S12XD 处理器概述

在内核模块级，Freescale MC9S12XD 系列微控制器具有相同的体系结构。一些产品的功能可能会多一些，并根据引脚数量的不同，封装也不尽相同。图 12-1 给出了 Freescale MC9S12XD 系列微控制器的基本功能框图，图 12-2 给出了 S12XD 系列微控制器之间的性能比较。

S12XD 系列微控制器具有高性能的 XGATE 协处理器模块，采用增强的直接存储器存取 (DMA) 功能，这种并行处理基于 RISC 的模块，减轻了 CPU 的负担，能够加快数据处理以及外设模块、RAM 和 I/O 端口之间的数据传输。XGATE 可以为 CPU 增加高达 80MIPS 的性能，从而可以提高产品的创新性和工作效率。

S12XD 系列微控制器的主要特征有：

- HCS12X 内核；
- 16 位的 HCS12XCPU；
- 与 HCS12 指令集的向上兼容性；
- 中断堆栈和程序员模型与 HCS12 相同；
- 指令队列；
- 增强的变址寻址；
- 增强的指令集；
- EBI (external bus interface, 外部总线接口)；

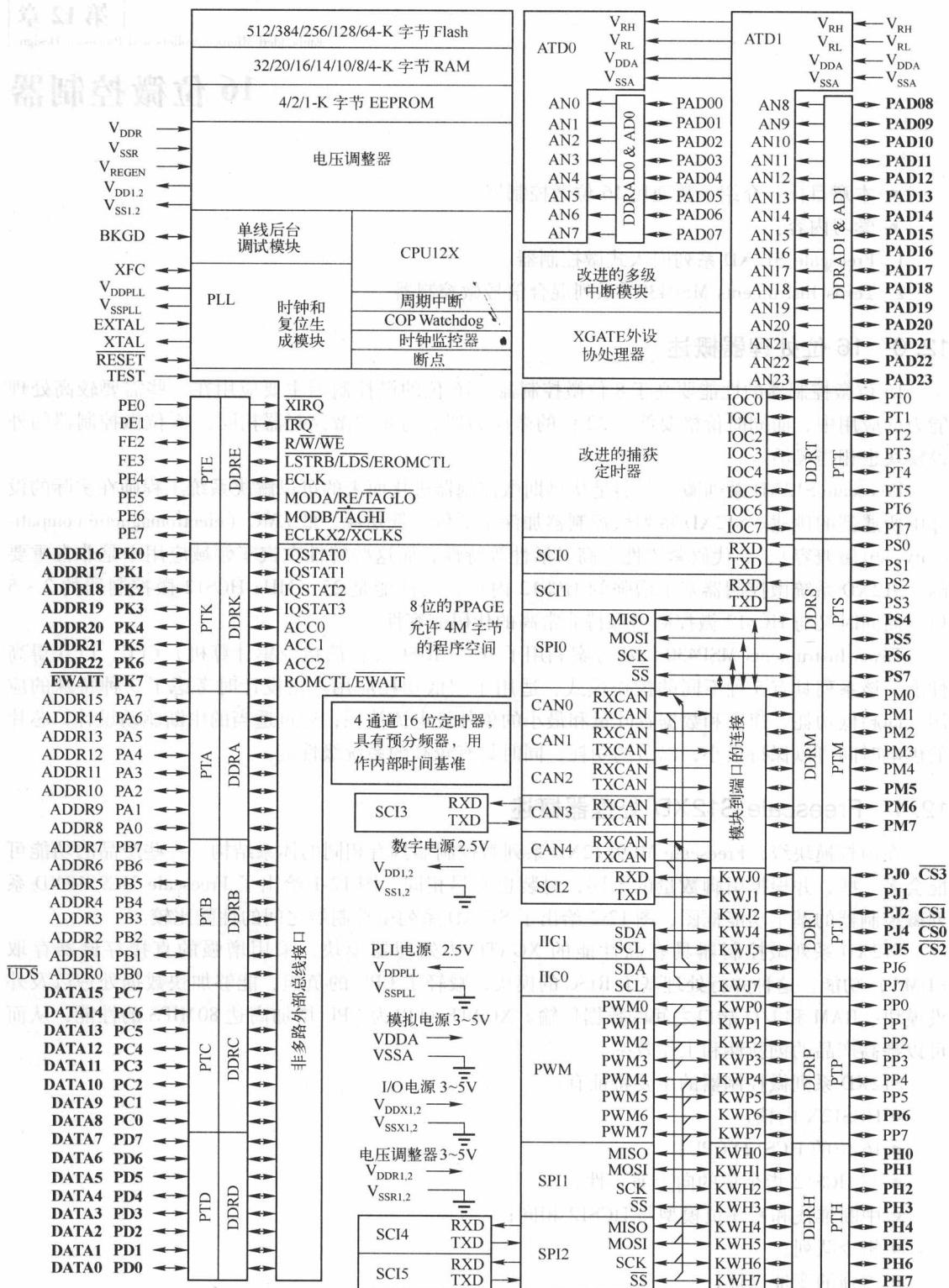


图 12-1 S12XD 系列微控制器功能框图 (Freescale 2007 版权许可)

属性	S12XD – new! (XB 和 XD 系列)	S12X 核	S12XS – new (XS 系列)	S12 核
总线速度	40 MHz	50 MHz	40 MHz	25 MHz
CPU	16 位 CPU12XV1	16 位 CPU12XV2	16 位 CPU12XV2	16 位 CPU12
调试	BDM (功能增强, 支持全局页访问) DBG 调试器监视 CPU XGATE 总线和 4 个比较器	BDM (功能增强, 支持全局页访问) DBG 调试器监视 CPU XGATE 总线和 4 个比较器	BDM (功能增强, 支持全局页访问) DBG 调试器监视 CPU	BDM (单线背景调试)
系统保护	低电压检测/中断	内存保护单元 (MPU) 对非正常访存进行保护 该系列的所有处理器都具有低电压检测/中断故障纠正代码	该系列的所有处理器都具有低电压检测/中断故障纠正代码	低电压检测/中断
振荡器选择	循环控制的或者增强的最大频率偏移皮尔斯电路, 用于动态控制输出振幅的增益, 以降低谐波失真 低功耗和很好的抗噪能力 消除偏压电阻	循环控制的或者增强的最大频率偏移皮尔斯电路, 用于动态控制输出振幅的增益, 以降低谐波失真 低功耗和很好的抗噪能力 消除偏压电阻	循环控制的或者增强的最大频率偏移皮尔斯电路, 用于动态控制输出振幅的增益, 以降低谐波失真 低功耗和很好的抗噪能力 消除偏压电阻	最大频率偏移皮尔斯的考毕兹
时钟	锁相环 (PLL)	锁相环增强电路, 不需要外部部件	锁相环增强电路, 不需要外部部件	锁相环 (PLL)
ADC	8/10 位分辨率, 7 $\mu$ s 转换时间	8/10/12 位分辨率, 2.12 $\mu$ s 转换时间	8/10/12 位分辨率, 2.12 $\mu$ s 转换时间	8/10 位分辨率, 7 $\mu$ s 转换时间
EEPROM	小块的 Flash, 模仿 EEPROM 的功能	仿真的 EEPROM, 具有数据 Flash 和 RAM 缓冲区	小块的 Flash, 模仿 EEPROM 的功能	小块的 Flash, 模仿 EEPROM 的功能
定时器	增强的捕获时间 (ECT) 定时器 (TIM) 定时中断定时器 (PIT)	增强的捕获时间 (ECT) 定时器 (TIM) 定时中断定时器 (PIT)	增强的捕获时间 (ECT) 定时器 (TIM) 定时中断定时器 (PIT)	增强的捕获时间 (ECT) 定时器 (TIM)
XGATE	XGATE 可编程高性能 I/O 协处理器模块 (最高可达 80MIPS RISC 性能)	XGATE 可编程高性能 I/O 协处理器模块 (最高可达 80MIPS RISC 性能)	不适用	不适用
中断嵌套	增强的中断模块, 共有 8 级	增强的中断模块, 共有 8 级	增强的中断模块, 共有 8 级	标准中断模块

图 12-2 S12XD 系列微控制器的比较 (Freescale 2007 版权许可)

- MMC (module mapping control, 模块映射控制);
- INT (interrupt controller, 中断控制);
- DBG (debug module to monitor, 监视 HCS12X CPU 和 XGATE 总线行为的调试模块);
- BDM (background debug mode, 背景调试模式)。

### 12.1.1 XGATE 概述

HCS12X 系列 16 位微控制器 (MCU) 中的 XGATE 模块具有高灵活性、高性能和低成本的并

行处理特点。XGATE 模块是一种外围协处理器，可以进行自动的高速数据处理，用于在 MCU 的外设和内部 RAM 和 I/O 端口之间高速传送数据；具有内嵌的 RISC 内核，能够对传送的数据进行预处理，并执行复杂的通信协议。

使用 XGATE 模块的目的是通过降低 S12X CPU 的中断处理负担，提高 MCU 的数据吞吐量。XGATE 模块以高达 80 MHz 的速度与 S12X CPU 并行运行，可以用 C 代码进行编程。XGATE 处理器功能是完全与 S12X CPU 集成在一起的，如图 12-3 所示。

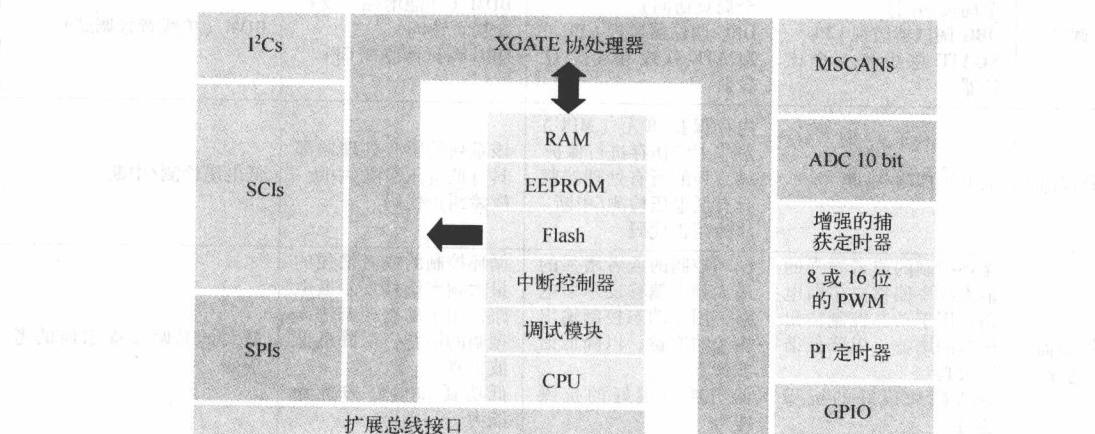


图 12-3 XGATE 集成 (Freescale 2007 版权许可)

XGATE 模块的功能特点有：

- 优化的 16 位 RISC 内核，用于数据处理；
- 内核之间采用硬件信号量安全的共享数据；
- 中断驱动的操作；
- 最多 112 个 XGATE 通道；
- 104 个硬件触发通道；
- 8 个软件触发通道；
- 在 XGATE 结束数据传输之后能够触发 S12X\_CPU 中断；
- 具有可以进行高速数据处理的筒形移位器；
- 采取智能内存访问保护，以防止与 CPU12 和 XGATE 的访问冲突。

图 12-4 给出了 XGATE 功能与 CAN 子系统之间的接口。图中表明 CAN 代码的内存映射由 XGATE RISC 处理器完成，这样就可以使 XGATE 指令的执行独立于 S12X\_CPU。

XGATE 功能的主要优势包括：

- 先进的、易于使用的中断处理外设；
- 可编程的“智能”DMA；
- 以低成本实现高达 80 MHz 的 CPU 并行处理频率；
- 所有外设和 RAM 之间的数据传输；
- 可以访问 30K 的 Flash (只读)；
- 当与 MSCAN 一起使用时，表现出非常灵活的完全 CAN 邮箱和滤波器管理功能；
- 高效的 multi-LIN 主设备功能；
- 易于采用标准 C 进行编程，编译器和调试器都集成在 S12X 工具套件中；
- 单背景调试模式 (BDM)，可以在全速执行的情况下同时调试 S12X\_CPU 和 XGATE。

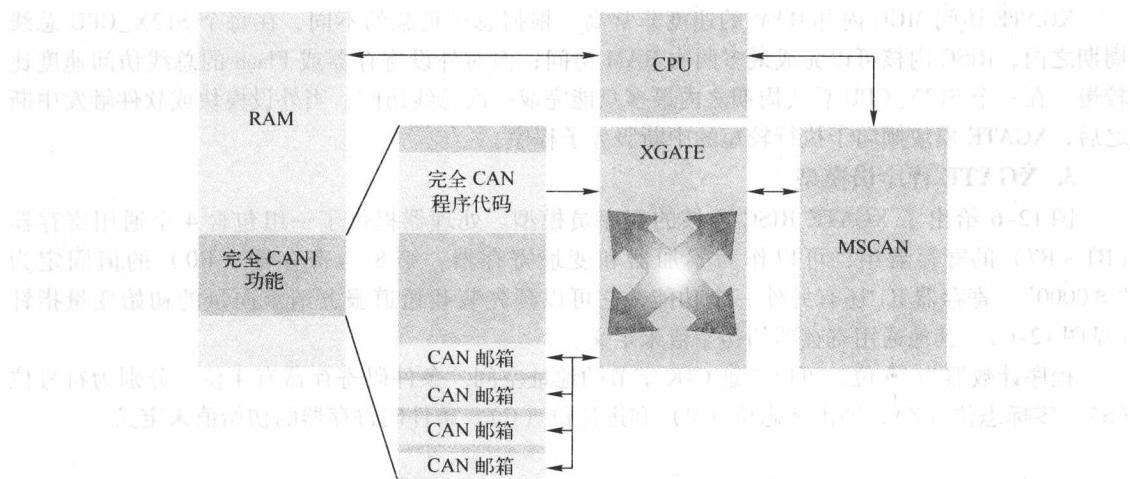


图 12-4 XGATE 与 CAN 的接口 (Freescale 2007 版权许可)

## 1. XGATE 模块

XGATE 模块的内核是一个 RISC 处理器，可以访问 MCU 的内部存储器和外设。被 XGATE 请求触发之前，RISC 处理器一直处于空闲状态；被触发后，该处理器执行与请求相对应的代码序列，并在完成处理后向 S12X\_CPU 发出相应的中断。完成处理之前，处理器所执行的代码序列是不可中断的。只有当当前面的代码序列执行完，并且 RISC 内核重新变为空闲之后，新的 XGATE 请求才能被响应。

XGATE 模块还提供了一组硬件信号量，用于在模块与 S12X\_CPU 共享 RAM 地址或外设的情况下确保数据一致。下面详细介绍 XGATE 模块的各组成部分。

## 2. XGATE RISC 内核

图 12-5 是 XGATE 模块的功能框图。RISC 内核是一个 16 位的处理器，它的指令集非常适用于数据传输、位操作和简单的算术运算，可以访问 MCU 的内部存储器和外设，同时不会阻塞 S12X\_CPU1 对这些设备的访问。当 S12X\_CPU 和 RISC 内核访问同一资源时，XGATE 模块将会先停止 RISC 的操作，直到该资源再次空闲为止。

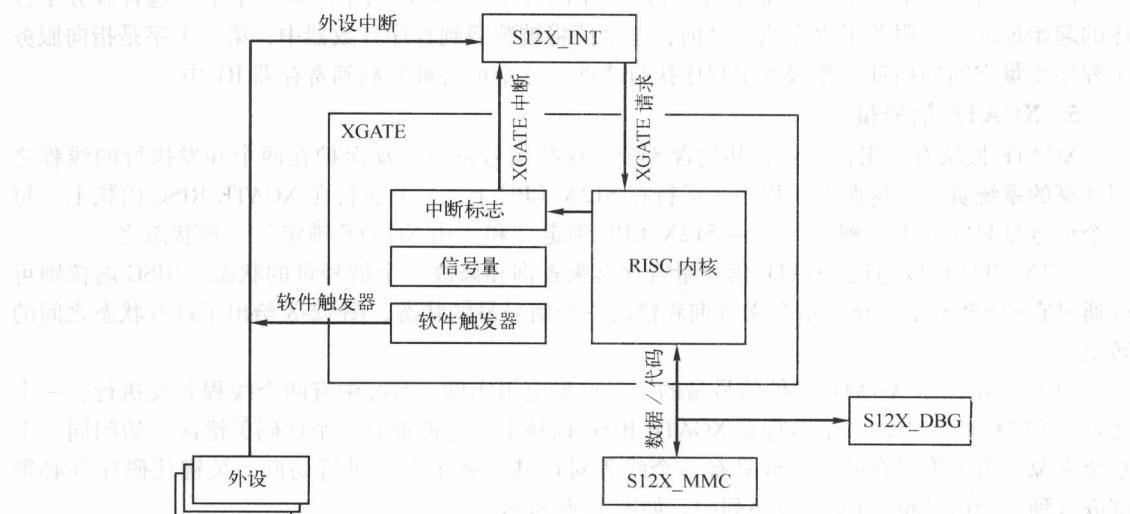


图 12-5 XGATE 框图 (Freescale 2007 版权许可)

XGATE 访问 MCU 内部 RAM 的速度非常高。根据总线负载的不同，在每个 S12X\_CPU 总线周期之内，RISC 内核可以完成最多两次 RAM 访问；但对外设寄存器或 Flash 的总线访问速度比较慢，在一个 S12X\_CPU 总线周期之内最多只能完成一次总线访问。当外设模块或软件触发中断之后，XGATE 模块倾向于执行较短的中断服务子程序。

### 3. XGATE 程序员模型

图 12-6 给出了 XGATE RISC 内核的程序员模型。处理器提供了一组包含 4 个通用寄存器 (R1 ~ R7) 的寄存器组，可以作为累加器和变址寄存器。第 8 个寄存器 (R0) 的值固定为“\$0000”。寄存器 R1 还有另外一个功能，它可以预先装载通道服务请求向量的初始变量指针（见图 12-6）。其他通用寄存器的初始值未定义。

程序计数器为 16 位，可以寻址 64K 字节的地址空间。条件码寄存器有 4 位，分别为符号位 (S)、零标志位 (Z)、溢出标志位 (V) 和进位位 (C)。条件码寄存器的初始值未定义。

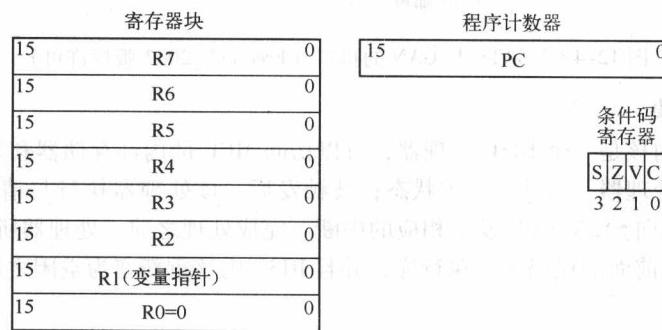


图 12-6 XGATE 程序员模型 (Freescale 2007 版权许可)

### 4. XGATE 存储器映射

XGATE 的 RISC 核可以访问 64K 字节的地址空间。存储器块在地址空间中的分配由芯片级确定。XGATE 向量块为每一个 XGATE 通道分配一个起始地址和一个变量指针，各个通道地址空间在 XGATE 内存中的映射可以通过 XGVBR 寄存器进行调整。

图 12-7 给出了向量块的分布情况。每一个向量有两个 16 位的字。第一个字中包含服务子程序的起始地址，在服务子程序执行之前，这个值将被装载到程序计数器中；第二个字是指向服务子程序变量空间的指针，在服务子程序执行之前，这个值将被装载到寄存器 R1 中。

### 5. XGATE 信号量

XGATE 模块有一组信号量，共包含 8 个。这些信号量可自动保护在两个并发执行的线程之间共享的系统资源，这两个线程一个运行在 S12X\_CPU 上，一个运行在 XGATE RISC 内核上。每一个信号量只能处于“解锁”、“由 S12X\_CPU 锁定”和“由 XGATE 锁定”三种状态之一。

S12X\_CPU 可以通过 XGATE 信号量寄存器来查询和修改一个信号量的状态。RISC 内核则可以通过它的 SSEM 和 CSEM 指令来查询和修改一个信号量的状态，图 12-8 给出了这些状态之间的转换。

图 12-9 给出了 XGATE 硬件信号量的一个典型应用实例。系统中有两个线程并发执行。一个运行在 S12X\_CPU 上，一个运行在 XGATE RISC 内核上。它们都有一个代码关键区，访问同一个系统资源。为了确保在同一时刻只有一个线程对该共享系统资源进行访问，关键代码序列必须被嵌入到一个信号量加锁/释放序列中，如图 12-9 所示。

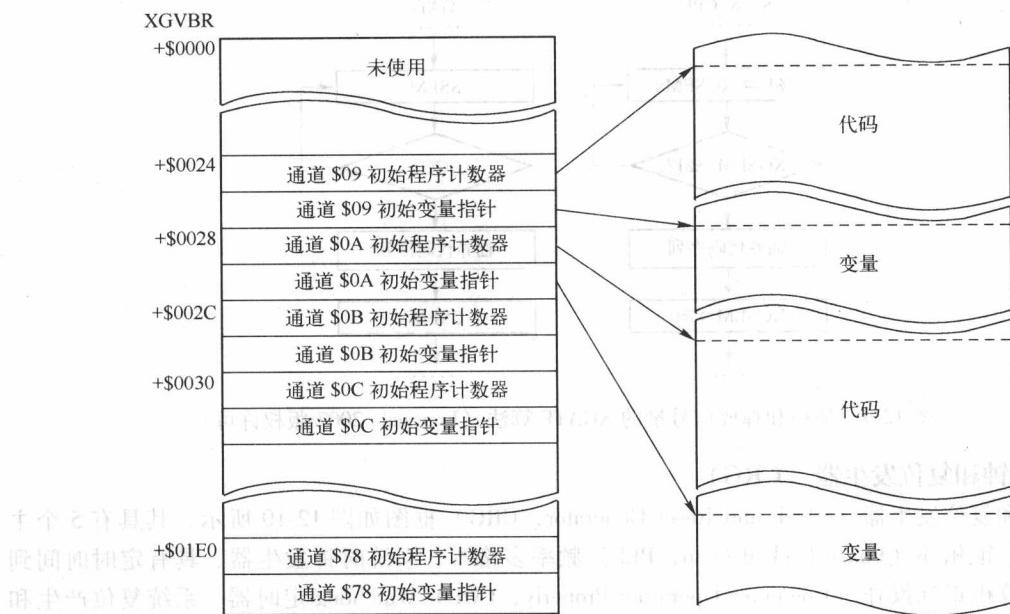


图 12-7 XGATE 向量块 (Freescale 2007 版权许可)

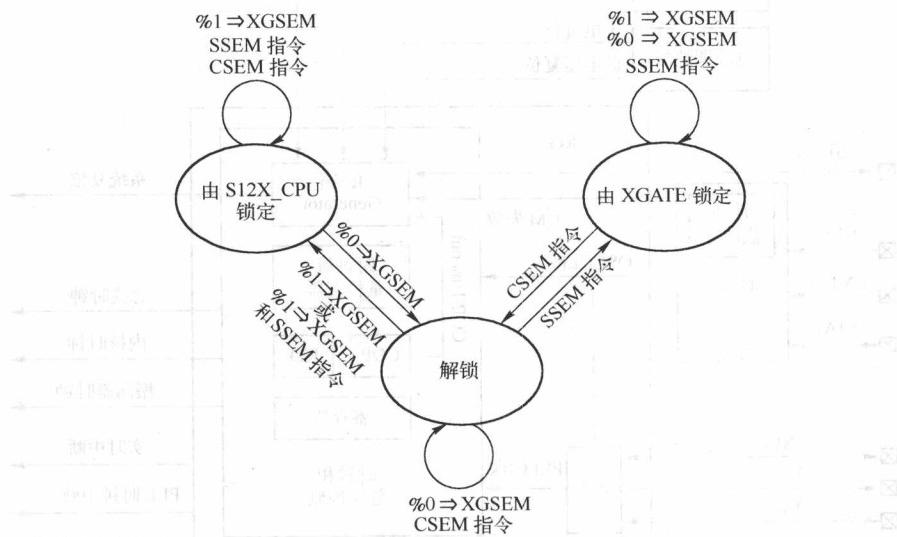


图 12-8 XGATE 信号量状态转换 (Freescale 2007 版权许可)

## 6. XGATE 操作模式

S12X 微控制器有 4 种运行模式，分别是运行模式、等待模式、停止模式和冻结模式（BDM 处于运行状态）。XGATE 可以工作在运行模式、停止模式和冻结模式中，当 XGATE 模块处于停止模式时，时钟将自动停止；在冻结模式下，根据模块的配置，XGATE 模块的所有时钟将停止。

### 12.1.2 时钟

系统共有时钟和复位产生功能模块和皮尔斯振荡器功能模块两种时钟功能。

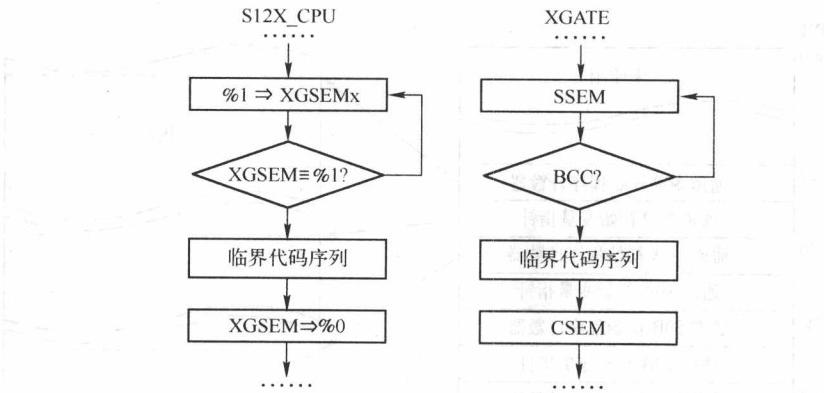


图 12-9 加锁和释放信号量的 XGATE 算法 (Freescale 2007 版权许可)

### 1. 时钟和复位发生器 (CRG)

时钟和复位发生器 (Clock and Reset Generator, CRG) 框图如图 12-10 所示，其具有 5 个主要的特点：锁相环 (Phase Locked Loop, PLL) 频率多路器、系统时钟发生器、具有定时时间到功能的计算机正确操作 (Computer Operating Properly, COP) Watchdog 定时器、系统复位产生和实时中断 (Real-Time Interrupt, RTI)。

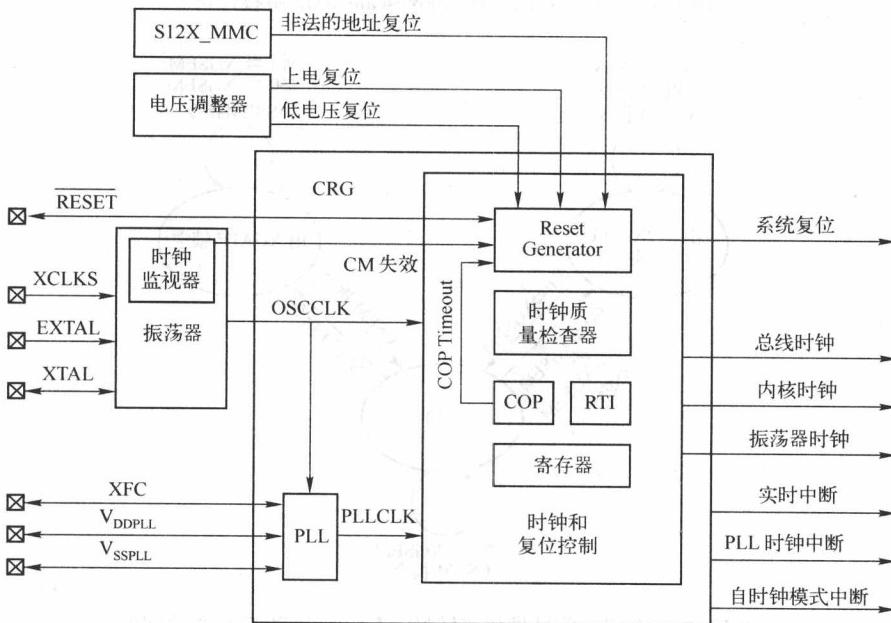


图 12-10 CRG 框图 (Freescale 2007 版权许可)

### 2. 皮尔斯振荡器 (XOSC)

皮尔斯振荡器 (XOSC) 模块可以产生高可靠性、低噪声和低功耗的时钟源，框图如图 12-11 所示。该振荡器模块由 VDDPLL 供电 (2.5V)，需要的外部模块的数量最小；在使用典型的晶体振荡器时，具有最佳的启动裕量。

XOSC 的电路可以自动控制输出振幅中的电流增加，这样就确保了信号具有较低的谐波畸变、低功耗和较好的噪声免疫性，基本的特点包括：

- 由于输入的滞后作用，具有高的噪声免疫性；
- 较低的RF散射，自动限制输出摆幅；
- 跨导( $g_m$ )大小可以进行调整，在使用典型的晶体振荡器时，具有最佳的启动裕量；
- 自动增益控制，可以不使用外部限流电阻；
- 使用集成电路，外部可以不用偏压电阻；
- 低功耗；
- 最低工作电压为2.5V；
- 振幅控制限制功率；
- 时钟监视。

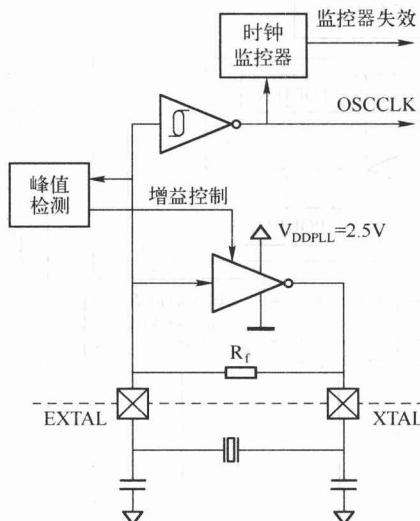


图12-11 皮尔斯振荡器(XOSC)框图(Freescale 2007版权许可)

### 12.1.3 模/数转换器(ATD)

ATD10B8C是一个8通道、10位、复用输入的连续逼近模数转换器，如图12-12所示，MC9S12XD还包括一个16通道的模数转换器(ATD10B16CV4)。8位模数转换器的主要特性有：

- 8/10位分辨率；
- 7μs、10位单转换时间；
- 采样缓冲放大器；
- 可编程采样时间；
- 左/右对齐的有符号/无符号数据结果；
- 外部触发控制；
- 转换结束产生中断；
- 用于16个模拟输入通道的模拟输入多路器；
- 模/数输入引脚复用；
- 1~16位的转换序列长度；
- 连续转换模式；
- 多通道扫描；
- 可以对任一AD通道或附加的4个外部触发输入的任意一个配置外部触发功能，附加的4

一个触发输入可以是片内的，也可以是片外的；

- 可配置通道回路位置（当在一个序列中转换多个通道时）。
- 是否对一个通道或多个通道执行一次或连续的转换，可以通过软件编程进行选择。

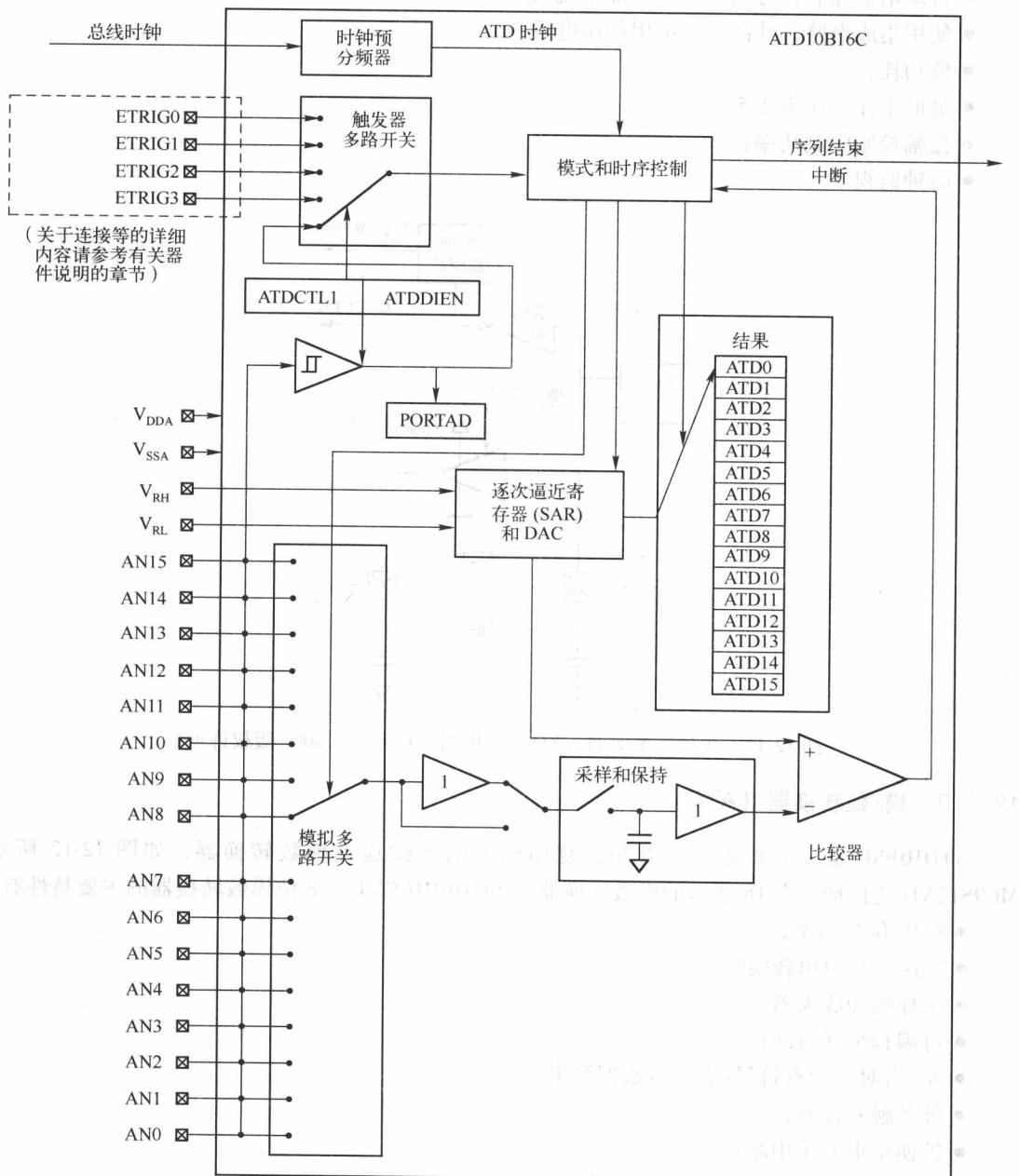


图 12-12 16 通道模数转换器 (Freescale 2007 版权许可)

#### 12.1.4 增强型捕捉定时器 (ECT)

HCS12 增强型捕捉定时器 (Enhanced Capture Timer, ECT) 模块对 HCS12 标准定时器模块的

功能进行了改进，如图 12-13 所示，增加的功能可以扩大应用领域，尤其是对于汽车 ABS 应用。该设计说明描述了标准的定时器以及附加的功能。

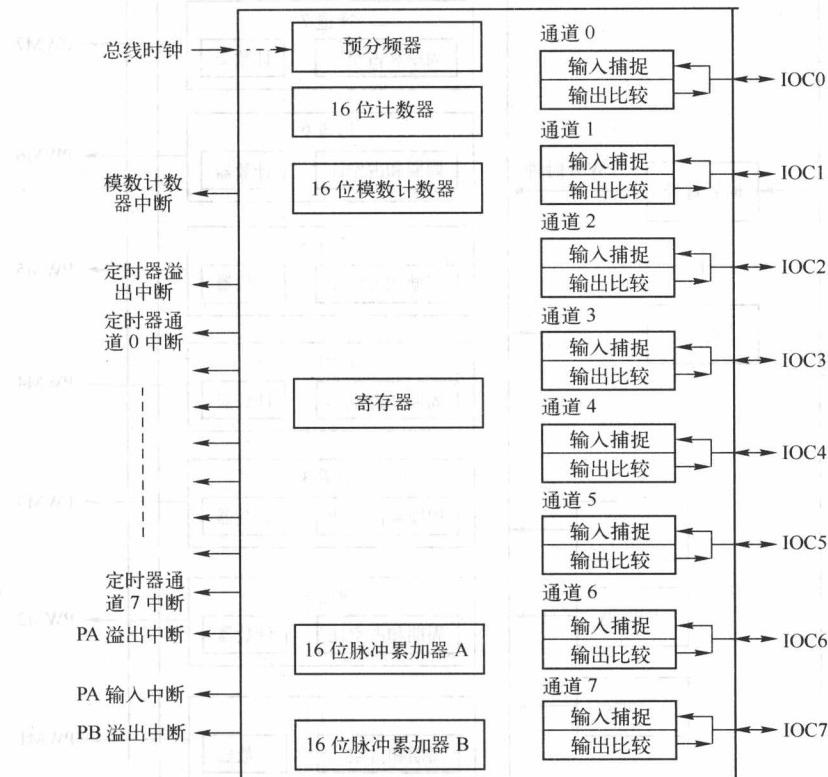


图 12-13 增强型捕捉定时器 (Freescale 2007 版权许可)

标准的定时器包括 1 个 16 位的、软件可编程的计数器，该计数器由预分频器驱动。该定时器可以有多种应用，包括输入波形测量的同时产生输出波形。脉冲宽度可以从几微妙到几秒不等。在一个时钟周期中，可以对计数器寄存器或者输入捕捉/输出比较寄存器进行一次完整的访问。如果对所有这些寄存器高低字节分开访问，与对它们进行按字访问相比，所获得的结果不同。

### 特点

- 4 个输入捕捉通道有一个 16 位的缓冲寄存器。
- 4 个 8 位的脉冲累加器，具有 8 位的缓冲寄存器和 4 缓冲 IC 通道。与 16 位的脉冲累加器一样，是可以配置的。
- 具有 8 位预分频器的 16 位模数逐减计数器。
- 4 个用户可选的延迟计数器，用于提高输入的抗噪声特性。

## 12.1.5 脉宽调制 (PWM)

脉宽调制 (PWM) 具有一些基本的特性，包括置中对齐的输出模式和 4 个可用的时钟源，如图 12-14 所示。PWM 模块具有 8 个通道，对于每个通道的居左对齐和居中对齐的输出都可以单独控制。

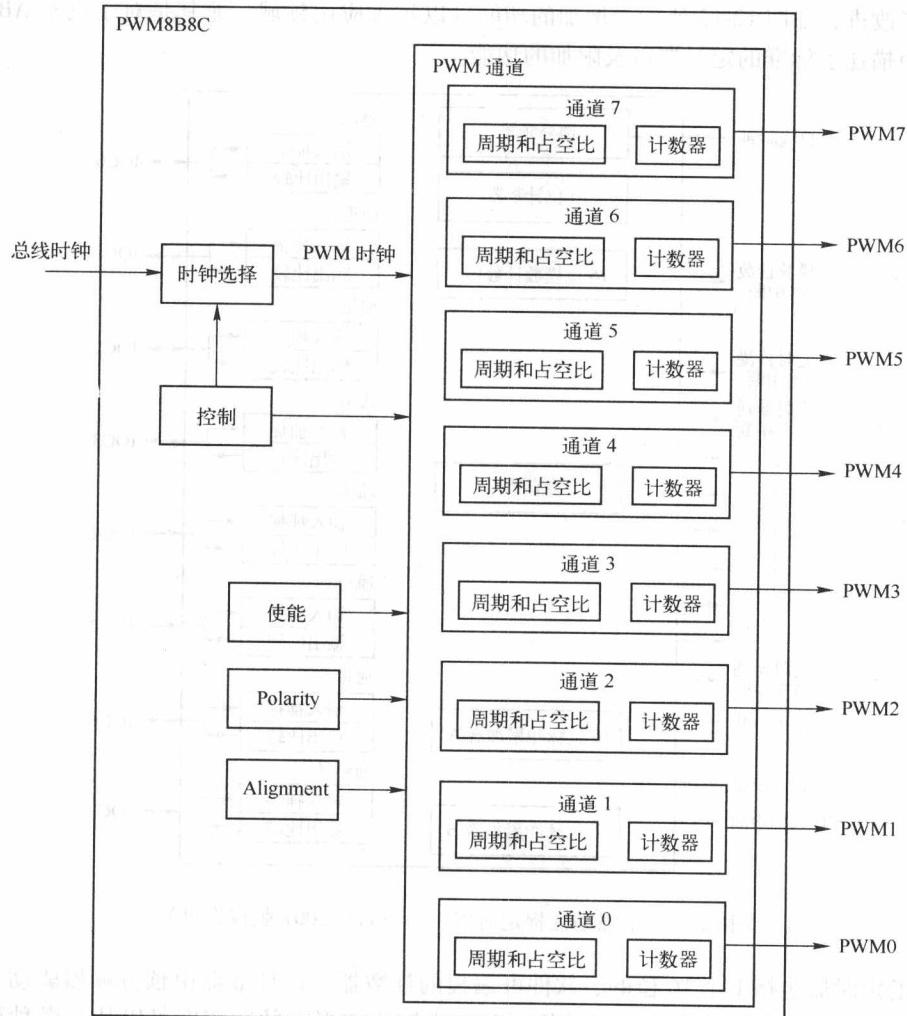


图 12-14 PWM 框图 (Freescale 2007 版权许可)

8 个通道中每一个的周期和占空比都是可编程的，并且都有一个专用的计数器。采用灵活的时钟选择方案，计数器可以有 4 个不同的时钟源。每个调制器可以产生独立的连续波形，并且可以通过软件选择 0 ~ 100% 的占空比。PWM 输出可以被编程为居左对齐的输出或居中对齐的输出。

### 特点

PWM 模块具有如下特点：

- 8 个独立的 PWM 通道，周期和占空比可编程。
- 每个 PWM 通道有专用的计数器。
- 每个通道都可编程为 PWM 使能或禁止。
- 每个通道都可以通过软件选择 PWM 占空脉冲极性。
- 周期和占空比是双缓冲的。在有效周期的最后（PWM 计数器到达 0）或者通道被禁止时，发生的改变会起作用。
- 每个通道可以单独编程为左对齐或中心对齐输出。

- 8个8位通道或者4个16位通道的PWM分辨率。
- 4个时钟源(A、B、SA和SB)可以提供很广的频率范围。
- 可编程的时钟选择逻辑。
- 紧急情况关闭。

### 12.1.6 I<sup>2</sup>C总线

I<sup>2</sup>C总线是一种双线的双向串行总线，可以在设备之间高效地交换数据（见图12-15）。由于采用双线制，I<sup>2</sup>C总线使得设备之间可以不需要大量的连接，因此不需要地址译码器。在几个短距离内的设备偶尔需要进行通信的情况下，可以使用该总线。当进行系统开发和扩展时，需要向总线上增加设备，此时使用该总线会比较灵活。

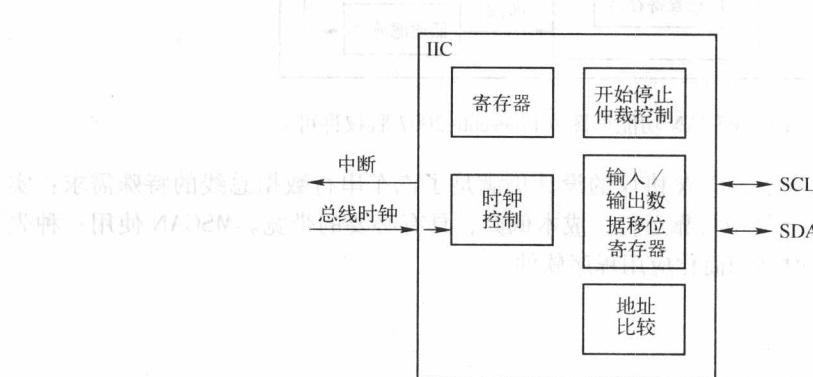


图12-15 I<sup>2</sup>C框图(Freescale 2007版权许可)

I<sup>2</sup>C接口的运行频率最高为100kbit/s，并设置最高的总线负载和时序。设备的波特率还可以更高，最高为时钟值/20，总线负载降低。其最长的通信距离和可以连接的设备数量，受到最高总线电容400pF的限制。

#### 特点

I<sup>2</sup>C模块的主要特点如下：

- 与I<sup>2</sup>C总线标准兼容。
- 多主设备操作。
- 256个不同的串行时钟频率软件可编程。
- 软件可选的应答位。
- 中断驱动的逐字节数据传输。
- 仲裁失败中断，可以自动从主模式转换到从模式。
- 调用地址识别中断。
- 起始和结束信号产生/检测。
- 重复起始信号产生。
- 应答位产生/检测。
- 总线忙检测。

### 12.1.7 CAN总线

Freescale的可伸缩的控制器局域网定义是基于MSCAN12定义的，MSCAN12是针对M68HC12微控制器系列，对MSCAN概念的特定实现。该模块（见图12-16）是一个通信控制器，

实现了 CAN 2.0A/B 协议，该协议在 Bosch 1991 年 9 月的说明书中进行了定义。

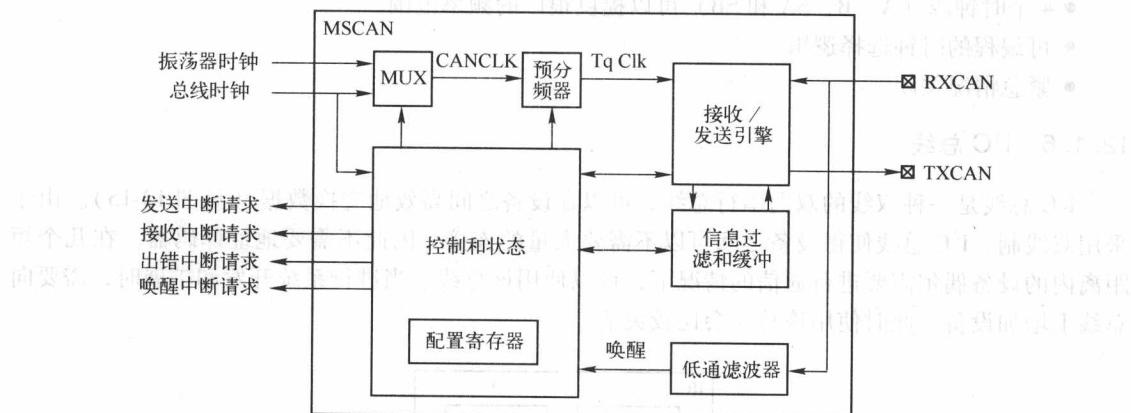


图 12-16 MSCAN 功能框图 (Freescale 2007 版权许可)

尽管并非专门用于汽车电子，CAN 协议的设计仍满足了汽车串行数据总线的特殊需求：实时处理，可以在汽车的 EMI 环境中可靠运行，成本低廉，具有必要的带宽。MSCAN 使用一种先进的缓冲管理，可以实现实时性和简化应用程序软件。

## 1. 特点

MSCAN 的基本特点如下：

- 实现 2.0A/B 版本的 CAN 协议。
- 标准的和扩展的数据结构。
- 0~8 字节的数据长度。
- 可编程的位频率最高可达 1 Mbit/s。
- 支持远程架构。
- 5 个具有 FIFO 存储模式的缓冲器。
- 3 个传输缓冲，实现“局部优先权”概念的内部优先次序安排。
- 灵活的、可屏蔽的标识符滤波器，支持两个全长（32 位）扩展的标识符滤波器，或者 4 个 16 位滤波器或 8 个 8 位的滤波器。
- 可编程的唤醒功能，具有集成的低通滤波器。
- 可编程的回送模式，支持自测操作。
- 可编程的只监听模式，用于监视 CAN 总线。
- 可编程的总线退出恢复功能。
- 对于所有的 CAN 接收器和发送器故障状态（警告、消极报错和总线退出），具有分开发信号和中断的能力。
- 可编程的 MSCAN 时钟源，可以是总线时钟或者振荡器时钟。
- 内部定时器，可以对接收和发生的消息加时间戳。
- 3 种低功耗模式——休眠、掉电和 MSCAN 使能。
- 对配置寄存器的全局初始化。

## 2. CAN 系统

典型的具有 MSCAN 的 CAN 系统如图 12-17 所示。每一个 CAN 栈都物理地通过收发器与 CAN 总线相连。收发器能够驱动 CAN 总线所需要的大电流，并具有电路包含功能，防止 CAN 缺

陷或者站缺陷。

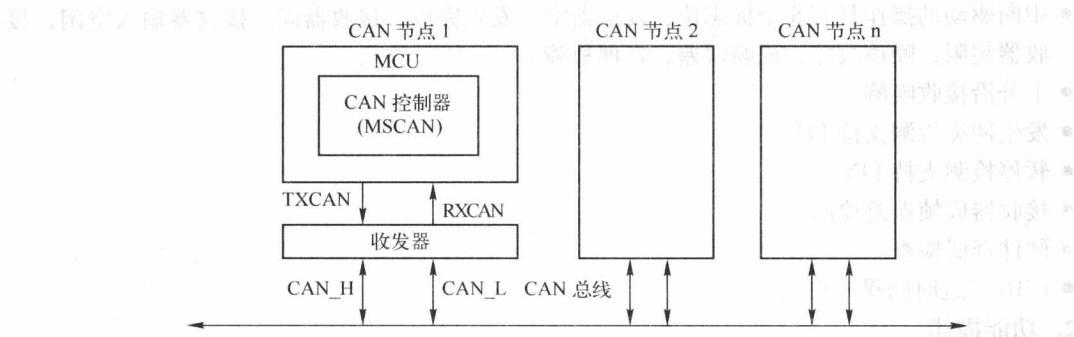


图 12-17 CAN 系统 (Freescale 2007 版权许可)

### 12.1.8 串行通信接口 (SCI)

串行通信接口 (Serial Communication Interface, SCI) 可以使设备与外设和其他 CPU 进行异步串行通信，框图如图 12-18 所示。SCI 非常灵活，可以被配置为多种串行接口。

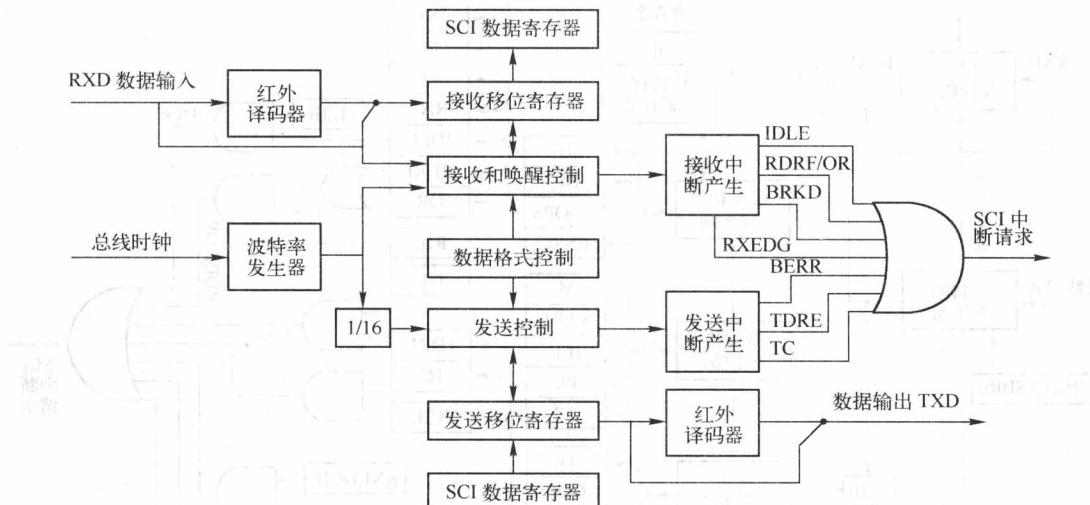


图 12-18 SCI 框图 (Freescale 2007 版权许可)

## 1. 特点

SCI 具有如下独特的特性：

- 选双工或单线操作。
- 标准的脉冲/间隔不归零 (non-return-to-zero, NRZ) 格式。
- 可选的 IrDA 1.4 归零翻转 (return-to-zero-inverted, RZI) 格式，脉冲宽度可编程。
- 13 位的波特率选择。
- 可编程的 8 位或 9 位数据格式。
- 可以对发送器和接收器单独使能。
- 发送器和接收器的极性可编程。
- 发送器输出奇偶可编程。

- 两种接收器唤醒方法：空闲线唤醒和地址标记唤醒。
- 中断驱动的操作具有 8 个标志位：发送器空，发生完成，接收器满，接收器输入空闲，接收器超限，噪声故障，成帧误差，奇偶故障。
- 上升沿接收唤醒。
- 发生冲突检测支持 LIN。
- 暂停检测支持 LIN。
- 接收器成帧误差检测。
- 硬件奇偶检查。
- 1/16 二进时间噪声检测。

## 2. 功能描述

本节将详细描述 SCI 块的功能，将在各个小节中，从终端用户的角度详细介绍设计的功能。

图 12-19 给出了 SCI 模块的详细结构。SCI 允许 CPU 和远程设备进行全双工、异步、串行通信，远程设备中也包括其他 CPU。尽管使用同一个波特率发生器，但是 SCI 的发送器和接收器独立运行。CPU 监视 SCI 的状态，向 SCI 写数据，等待发生，并处理接收的数据。

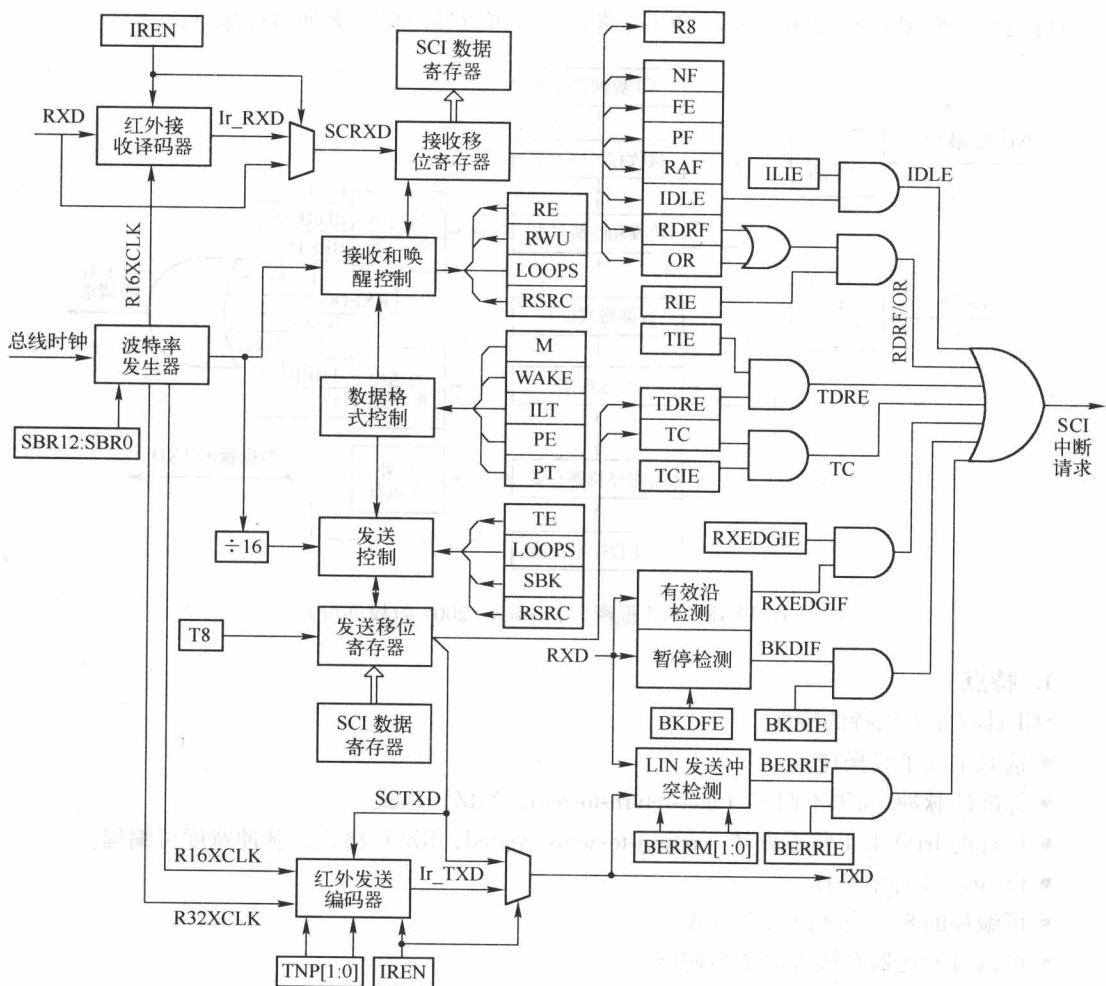


图 12-19 详细的 SCI 框图 (Freescale 2007 版权许可)

### (1) 红外线接口 (IrDA)

IrDA 可以向 IR LED 发送窄脉冲，并将它们转换为串行数据位，然后发送到 SCI。IrDA 物理层说明书定义了一个半双工红外线通信链接，用于交换数据。在完整的标准中，数据的传输速率高达 16Mbit/s。而在这里只支持 2.4Kbit/s 到 115.2Kbit/s 的速度。

红外线子单元共包含发生编码器和接收译码器两个主要的模块。SCI 发生串行数据，由红外线子单元进行编码，每一个 0 位都会发送一个窄脉冲。对于 1 位，不发送脉冲。

当接收数据时，用 IR 光电二极管检测 IR 脉冲，并通过 IR 接收译码器（来自外部的 MCU）将其转换为 CMOS 电位。然后，窄脉冲将由红外线子单元进行扩展，变为可以被 SCI 接收的串行数据流。发送脉冲和接收脉冲的极性可以进行翻转，这样可以与外部使用的有效脉冲的 IrDA 收发器单元建立直接的连接。

### (2) LIN 支持

该模块对局域网互联 (LIN) 协议提供了基本的支持。首先，这是一个暂停检测电路，使 LIN 软件能够很容易从到来的数据流中分辨出暂停字符。该模块还有一个附加功能，可以在位级检测到冲突，并可以消除推迟传输。

## 3. 数据格式

SCI 使用标准的 NRZ 脉冲间隔数据格式。当红外线可用时，如果 0 用光脉冲代替，1 保持为低，则 SCI 使用 RZI 数据格式，如图 12-20 所示。

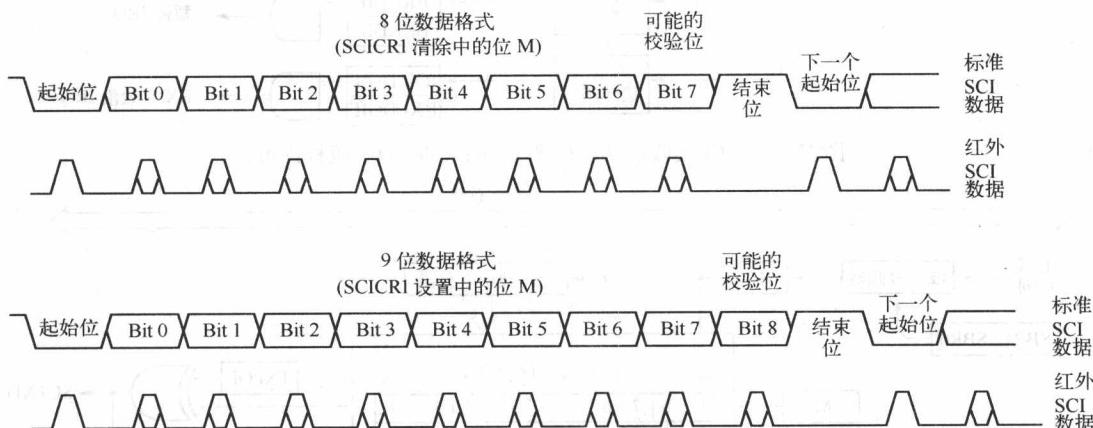


图 12-20 SCI 数据格式 (Freescale 2007 版权许可)

## 4. 接收器

SCI 接收器功能框图如图 12-21 所示。

## 5. 发送器

SCI 发送器功能框图如图 12-22 所示。

## 6. 波特率发生器

波特率发生器中有一个 13 位的系数计数器，可以从发送器和接收器获得波特率。写入到 SBR 12:SBR 0 位的 0~8191 之间的值，决定了总线时钟约数。波特率时钟与总线时钟进行同步，并驱动接收器。波特率时钟除以 16 后驱动发送器，接收器的采集速率为每单位时间采样 16 次。

图 12-23 给出了一些例子，以 25 MHz 的总线频率获得目标波特率。

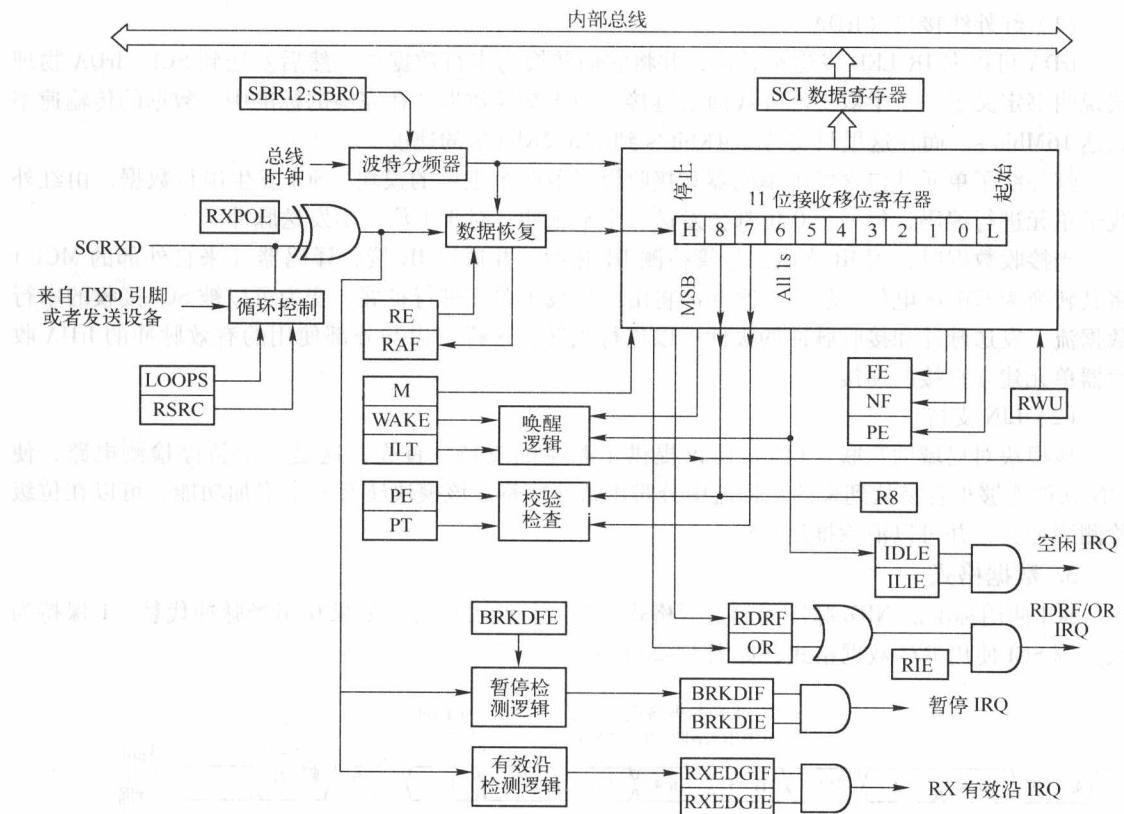


图 12-21 SCI 接收器功能框图 (Freescale 2007 版权许可)

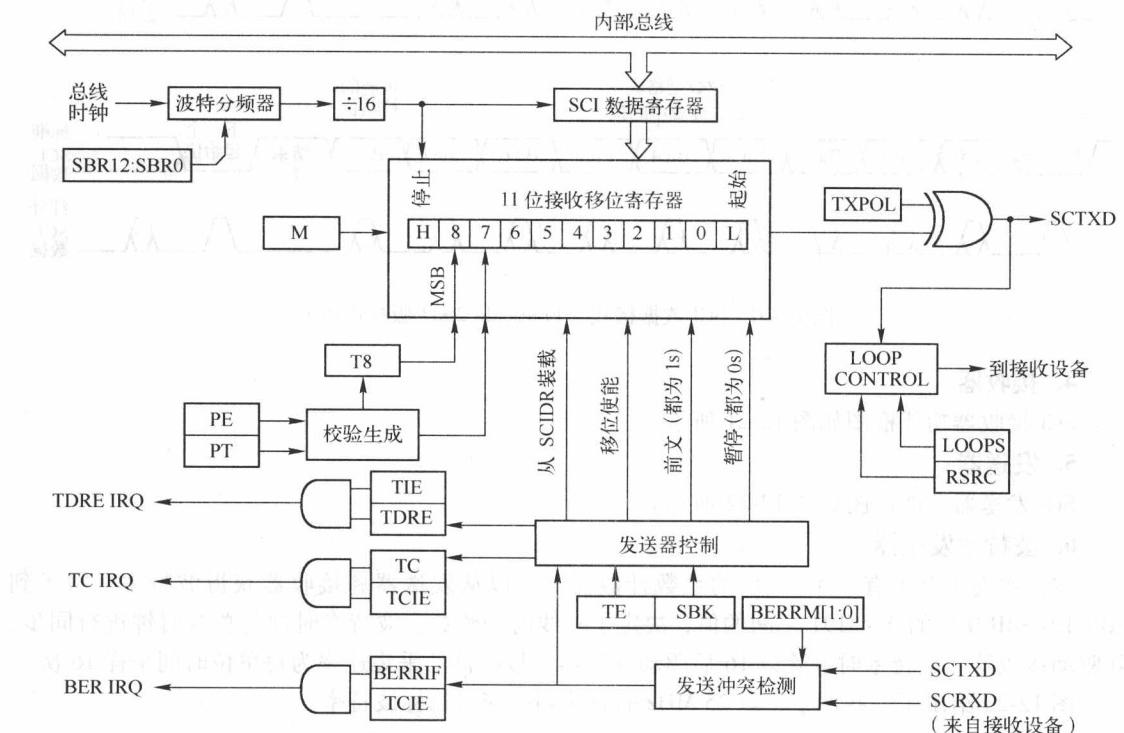


图 12-22 SCI 发送器功能框图 (Freescale 2007 版权许可)

位 SBR [12:]	接收设备 时钟 (Hz)	发送设备 时钟 (Hz)	目标 波特率	出错 (%)
41	609 756.1	38 109.8	38 400	.76
81	308 642.0	19 290.1	19 200	.47
163	153 374.2	9 585.9	9 600	.16
326	76 687.1	4 792.9	4 800	.15
651	38 402.5	2 400.2	2 400	.01
1 302	19 201.2	1 200.1	1 200	.01
2 604	9 600.6	600.0	600	.00
5 208	4 800	300.0	300	.00

图 12-23 波特率发生器表 (Freescale 2007 版权许可)

### 12.1.9 串行外围接口 (SPI)

MCU 和外设可以通过 SPI 模块进行双工、同步、串行通信，软件可以查询 SPI 状态标志位，SIP 的操作也可以是中断驱动的。

#### 1. 特点

SPI 具有以下特点：

- 主模式和从模式。
- 双向模式。
- 从选择输出。
- 具有 CPU 中断能力的模式故障标志。
- 双缓冲的数据寄存器。
- 极性和相位可编程的串行时钟。
- 在等待模式中控制 SPI 操作。

图 12-24 给出了 SPI 体系结构描述。SPI 的主要组成部分有状态寄存器、控制寄存器和数据寄存器、移位器逻辑、波特率发生器、主/从控制逻辑和端口控制逻辑。

SPI 模块具有如下标准的 4 个外部引脚：

- MOSI (Master Out/Slave In Pin) 主输出/从输入引脚。
- MISO (Master In/Slave Out Pin) 主输入/从输出引脚。
- SS (Slave Select Pin) 从选择引脚。
- SCK 串行时钟引脚。

#### 2. 功能描述

SPI 使 MCU 和外设之间能够进行双工、同步、串行的通信，如图 12-25 所示。软件可以查询 SPI 状态寄存器，SPI 操作还可以是中断驱动的。通过设置 SIP 控制寄存器 1 中的 SIP 使能 (SPE) 位，可以启用 SPI 系统。

SPI 系统的主要构成部分是 SPI 数据寄存器。主设备中的 8 位数据寄存器和从设备中的 8 位数据寄存器可以通过 MOSI 和 MISO 引脚连接在一起，形成一个分布式的 16 位寄存器。当执行数据传输操作时，该 16 位寄存器可以在主设备中 S 时钟的控制下串行移动 8 位，这样就可以实现主设备和从设备之间的数据交换。写入主设备 SPI 数据寄存器中的数据变成输出到从设备的数据；传输操作之后，从主设备 SPI 数据寄存器读取的数据，变成来自于从设备的输入数据。

### 主/从模式

当MSTR位被设置为1时，SPI工作在主模式下，只有主SPI模块才可以初始化传输过程。传输的第一步就是写SPI数据寄存器，如果移位寄存器为空，则数据立即被传送到移位寄存器。在串行时钟的控制下，数据开始移位输出到MOSI引脚。

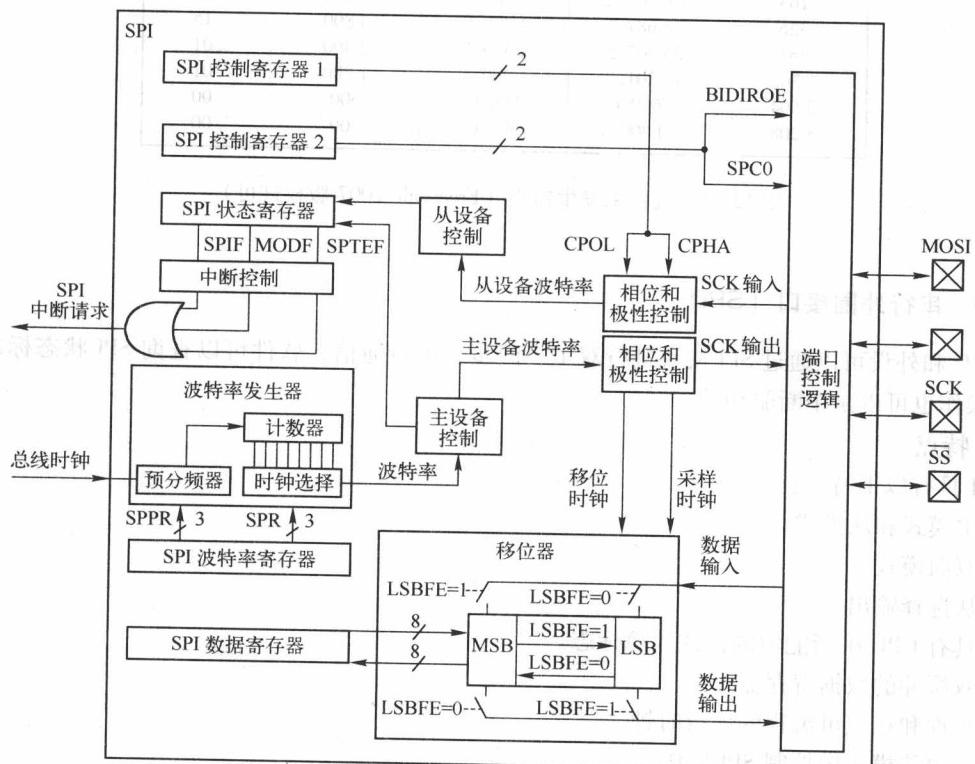


图 12-24 SPI 框图 (Freescale 2007 版权许可)

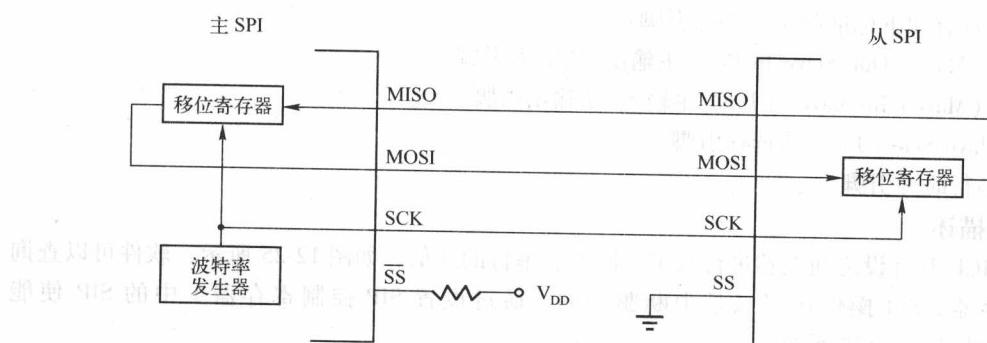


图 12-25 主/从传输模块框图 (Freescale 2007 版权许可)

尽管 SPI 可以以双工模式操作，一些 SPI 外设却只能以从模式接收 SPI 数据。对这些简单的设备来说，没有串行数据输出引脚。尽管主设备不可能从所有接收数据的从设备中获得返回信息，只要驱动系统从串行数据输出线的从设备不多于一个，就可以有多个从设备接收一个主设备发送来的同一个数据。

在SPI传输过程中，数据的发生（串行移位输出）和接收（串行移位输入）是同时进行的，串行时钟（SCK）对两个串行数据线信息的移位和采样进行同步。从设备选择性可以只选择一个从SPI设备，没有被选择的从设备不干扰SPI总线行为。在主SPI设备中，从选择线可以用于表示多主设备总线争议。

### 12.1.10 定时中断定时器（PIT）

定时中断定时器（Periodic Interrupt Timer, PIT）是一个24位定时器阵列，可以用于触发外设模块，或者用于周期性的中断，如图12-26所示。

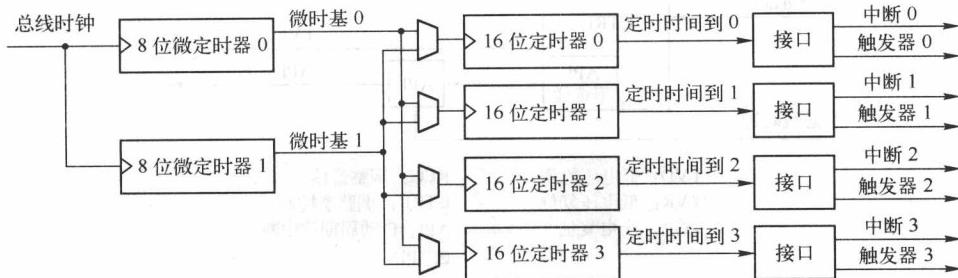


图12-26 PIT框图（Freescale 2007版权许可）

#### 特点

PIT具有如下特点：

- 4个定时器构成系数递减计数器，具有独立的定时周期。
- 定时周期可选，可以在1~224个总线时钟周期之间进行选择，定时周期等于m\*n个总线周期，其中 $1 \leq m \leq 56$ ,  $1 \leq n \leq 65536$ 。
- 可以分别设置使能的定时器。
- 4个定时时间到中断。
- 4个定时时间到触发输出信号，可用于触发外设模块。
- 定时器通道的启动可以互相对准。

### 12.1.11 电压调整器（VREG）

电压调整器（Voltage Regulator, VREG）模块是一个双输出的电压调整器，可以提供2个独立的2.5V（典型）电压，而输出电流根据需要会有所不同，输入电压范围为3.3~5V（典型）。VREG功能框图如图12-27所示。

#### 特点

VREG模块具有如下特点：

- 两个平行的、线性的电压调整器。
- 低电压检测（Low-voltage detect, LVD）功能，可以发出低电压中断（low-voltage interrupt, LVI）。
- 上电复位（POR）。
- 低电压复位（Low-voltage reset, LVR）。
- 自主周期中断（Autonomous periodical interrupt, API）。

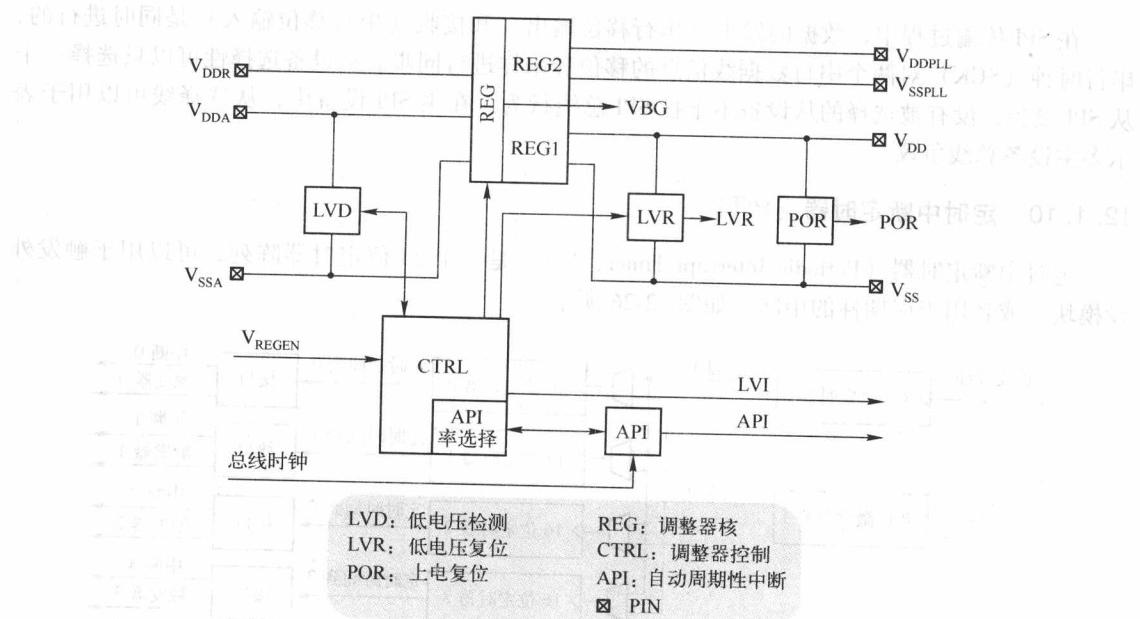


图 12-27 VREG 框图 (Freescale 2007 版权许可)

### 12.1.12 背景调试模块 (BDM)

背景调试模块是一个单线的、背景调试系统，用于减小 CPU 的干预，框图如图 12-28 所示，与 BDM 的所有接口都是通过 BKGD 引脚进行的。

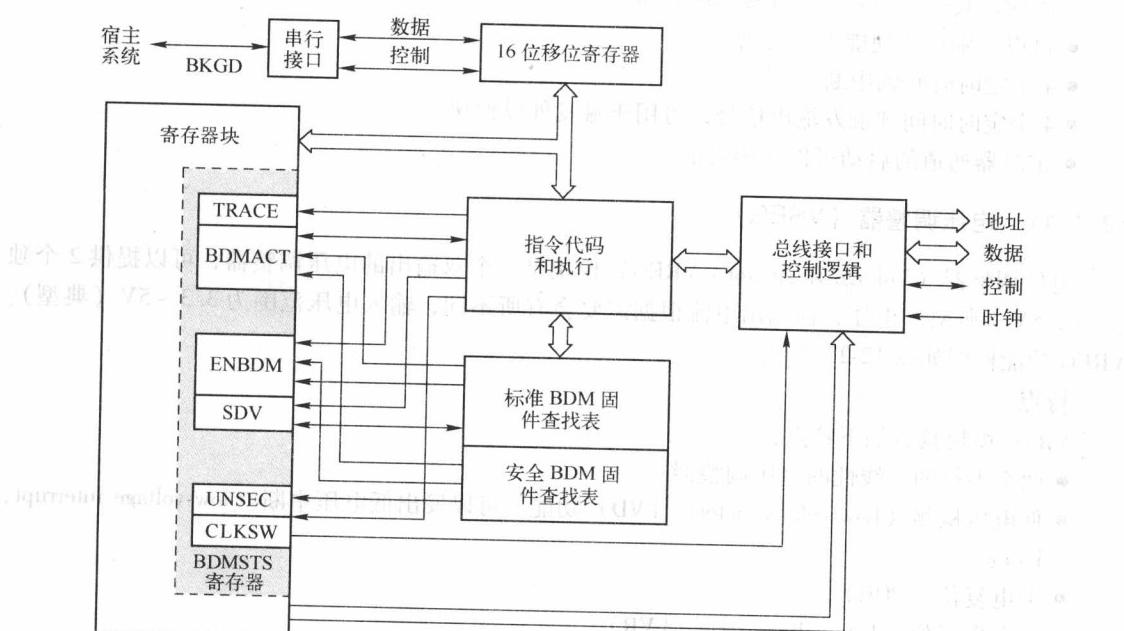


图 12-28 BDM 框图 (Freescale 2007 版权许可)

BDM 能够保持目标系统和主机的同步，而同时对于时钟频率具有更高的灵活性。这里包括一个 sync 信号来确定通信速率，还有一个确定操作何时结束的握手信号。

### 特点

BDM 具有如下特点：

- 与主开发系统进行单线通信。
- 能够保持时钟速率的高度灵活性。
- 确定通信速率的 SYNC 命令。
- GO\_UNTIL 命令。
- 硬件握手协议，以提高串行通信的效率。
- 在特殊的单芯片模式中从复位恢复为活跃状态。
- 9 个使用空闲周期的硬件命令，尽量减少 CPU 的干预。
- 不需要启动 BDM 的硬件命令。
- 14 个固件命令，从标准的 BDM 固件查找表开始执行。
- 在等待模式中对 BDM 操作进行软件控制。
- 软件可选择的时钟。
- 全局页访问功能。
- 可以运行在模拟模式下，但是复位之后处于无效状态。
- 在模拟模式中复位后，SLKSW 位被设置为有效。
- 在安全的情况下，如果 Flash 和 EEPROM 擦除测试失败，允许硬件命令在特殊的单芯片模式下访问寄存器空间。
- 可以从全局地址 0x7FFF0F 上的固件 ROM 读取系列 ID 号（HCS12X 的 ID 号为 0xC1）。
- 在系统进入停止模式之前，BDM 硬件命令是可选的（所有总线主设备都处于停止模式）。

### 12.1.13 中断模块 (XINT)

中断模块 (Interrupt Module, XINT) 将所有的系统异常请求优先权进行译码，并给 CPU 或者 XGATE 模块提供用于处理异常的有效向量，如图 12-29 所示。XINT 模块支持如下请求。

- I 位和 X 位可屏蔽的中断请求。
- 不可屏蔽的、未实现的操作码陷阱。
- 不可屏蔽的软件中断 (software interrupt, SWI) 或者背景调试模式请求。
- 假中断向量请求。
- 3 个系统复位向量请求。

每一个 I 位可屏蔽的中断请求可以具有 7 个优先级之一，支持灵活的优先级方案。对于需要由 CPU 处理的中断请求，优先级方案可以用于实现嵌套的中断功能，这样如果具有高优先级的中断正在处理中，来自于较低层次的中断可以自动被阻塞。需要由 XGATE 模块处理的中断请求不能嵌套，因为 XGATE 在处理过程中不能被打断。

#### 1. 特点

- 中断向量地址寄存器 (Interrupt vector base register, IVBR)。
- 一个假中断向量 (位于地址向量基址 1 + 0x0010)。

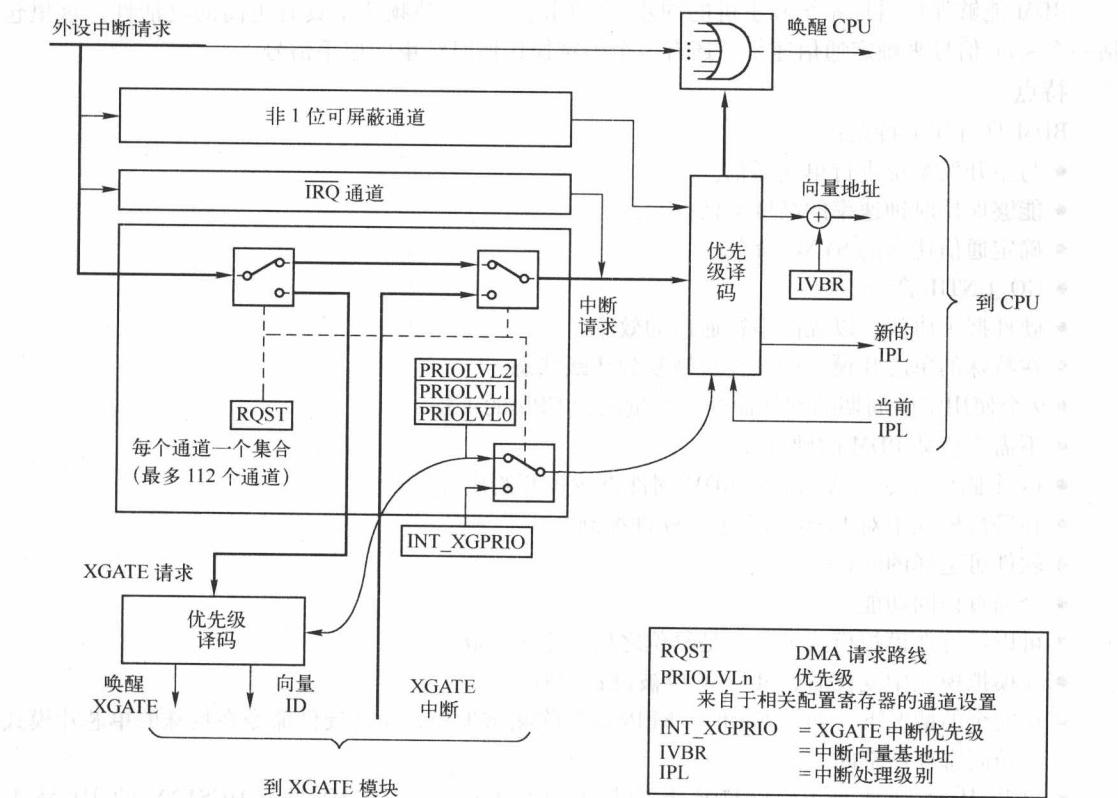


图 12-29 中断控制框图 (Freescale 2007 版权许可)

- 2 ~ 113 个 I 位可屏蔽的中断向量请求（位于地址向量基地址  $1 + 0x0011 \sim 0x00F2$ ）。
- 每一个 I 位可屏蔽的中断请求都具有一个可配置的优先级，可以配置为由 CPU 或者 XGATE 模块 2 进行处理。
- 根据优先级的不同，I 位可屏蔽的中断可以进行嵌套。
- 一个 X 位可屏蔽的中断向量请求（位于地址向量基地址  $1 + 0x00F4$ ）。
- 一个不可屏蔽的软件中断向量请求（SWI）或者背景调试模式向量请求（位于地址向量基地址  $1 + 0x00F6$ ）。
- 一个可屏蔽的未实现操作码陷阱（TRAP）向量（位于地址向量基地址  $1 + 0x00F8$ ）。
- 3 个系统复位向量（位于地址  $0xFFFFA \sim 0xFFFFE$ ）。
- 确定最高优先级 DMA 和中断向量请求，分别将向量传输给 XGATE 模块；或者当请求 CPU 服务时传输给总线。
- 即使在 X 中断被屏蔽的情况下，当适当的中断请求发生或者 XIRQ 发出后，将系统从停止模式或等待模式唤醒。
- 即使 CPU 仍然处于停止或等待模式，XGATE 也可以唤醒并执行代码。

## 2. 中断嵌套

中断请求优先级方案的实现，可以对 CPU 处理的 I 位可屏蔽的请求，实现基于优先级的中断请求嵌套。

I 位可屏蔽的中断请求可以被具有更高优先级的中断请求打断，这样可以同时具有最

7 个嵌套的 I 位可屏蔽中断请求（图 12-30 给出了一个例子，具有多达 3 个嵌套的中断请求）。

I 位可屏蔽的中断请求不可以被另外一个 I 位可屏蔽中断请求打断。为了使中断服务子程序 (ISR) 可以被打断，ISR 必须清除 CCR (CLI) 中的 I 位。将 I 位清除后，具有更高优先级的 I 位可屏蔽中断请求即可以打断当前的 ISR。

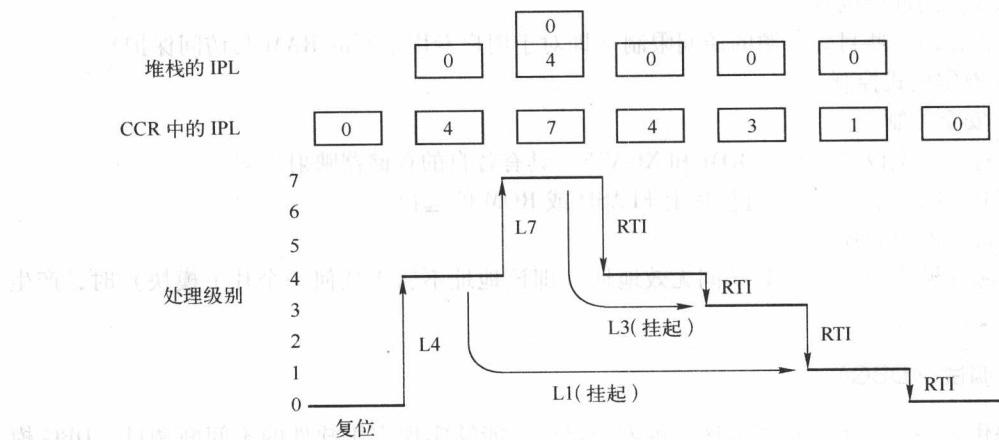


图 12-30 嵌套的中断实例 (Freescale 2007 版权许可)

可被打断的 I 位可屏蔽中断请求的 ISR 的执行顺序如下：

- 服务中断，即清除中断标志，复制数据。
- 执行指令 CLI 清除 CCR 中的 I 位，因此允许中断请求具有更高的优先级。
- 处理数据。
- 执行 RTI 指令 RTI，从中断返回。

#### 12.1.14 映射存储器控制 (MMC)

MMC (S12XMMCV3) 框图如图 12-31 所示。MMC 模块控制多主设备优先权访问、内部资源的选择和外部空间；内部总线，包括内部存储器和外设，都由该模块进行控制。每个主设备的局部地址空间都要被转换成全局地址空间。

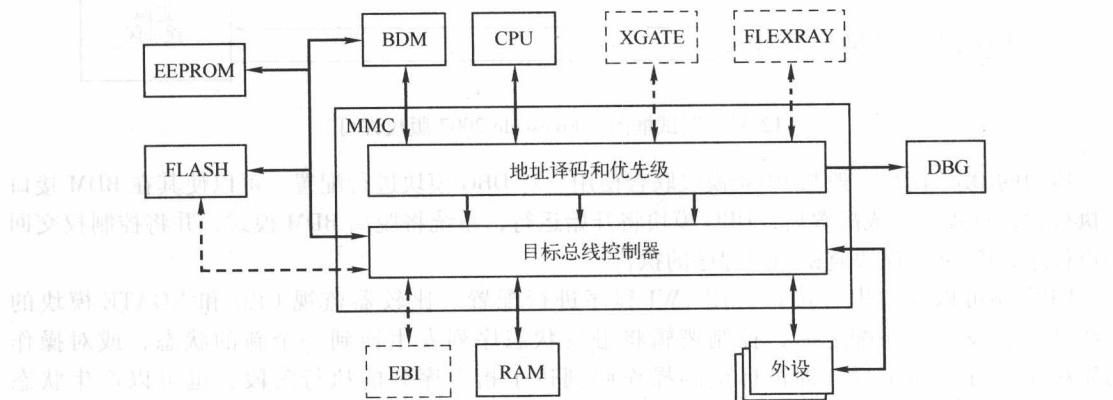


图 12-31 MMC 框图 (Freescale 2007 版权许可)

## 特点

MMC 的主要特点如下：

- 可以分页，支持全局的 8M 字节存储地址空间。
- 主设备 CPU、BDM 和 XGATE 之间的总线仲裁。
- 对不同的资源可以同时访问（内部的、外部的和外设）。
- 对总线访问冲突的裁决。
- 主设备控制一些目标资源的访问限制（即对于用户专用空间的 RAM 写访问保护）。
- MCU 操作模式控制。
- MCU 安全控制。
- 对于每一个主设备 CPU、BDM 和 XGATE，具有各自的存储器映射方案。
- 具有 ROM 控制位，可以进行片上 FLASH 或 ROM 的选择。
- 端口寄存器访问控制。
- 在单芯片模式中，当 CPU 访问无效地址（即该地址不属于任何一个片上模块）时，产生系统复位信号。

### 12.1.15 调试 (DBG)

DBG 提供了一个片上追踪缓冲区，触发很灵活，能够实现应用软件的不间断调试。DBG 模块适用于 HCS12X 16 位体系结构，可以对 CPU 和 XGATE 模块的操作进行调试，功能框图如图 12-32 所示。

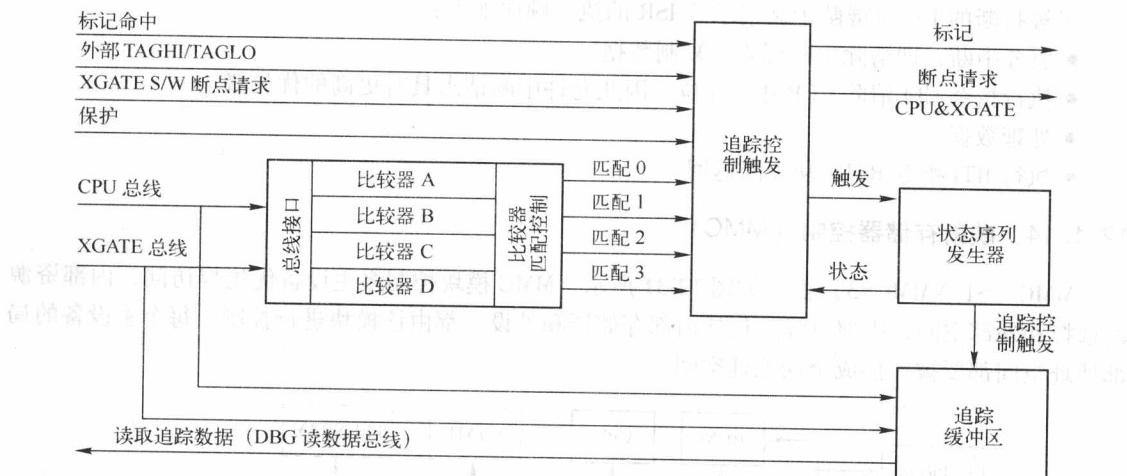


图 12-32 调试框图 (Freescale 2007 版权许可)

典型的 DBG 模块一般与 BDM 模块联合使用。对 DBG 模块进行配置，可以使其在 BDM 接口上执行调试过程。完成配置后，DBG 模块将开始运行，系统将脱离 BDM 模式，并将控制权交回用户程序，由 DBG 模块监视用户程序的执行。

DBG 还可以通过串行接口，用 SWI 程序进行配置。比较器监视 CPU 和 XGATE 模块的总线状态，发生了匹配之后，控制逻辑将触发状态序列发生器到一个新的状态，或对操作码做标记。标记命中后，即被标记的操作码到达了指令序列的执行阶段，也可以产生状态转换。

当向结束状态转换时，可以触发总线追踪和/或产生一个断点。相关的追踪和断点所引发的向结束状态的转换，可以由外界的 TAGHI 和 TAGLO 信号触发，这个过程与比较器匹配没有关系。上述的过程可以通过 XGATE 模块 S/W 断点请求，或者向 TRIG 控制位写相应的数值来完成。通过寄存器地址映射的 2 字节窗口可以查看跟踪缓冲区，并且可以通过标准的 16 位读字操作将其内容读出。当 MCU 系统被保护后，跟踪也被禁止。

## 特点

BDM 的主要特点如下：

- 4 个比较器（A、B、C 和 D）：
  - 比较器 A 和 C 对全地址和全 16 位数据总线进行比较。
  - 比较器 A 和 C 的主要特点是具有数据总线屏蔽寄存器。
  - 比较器 B 和 D 只对全地址总线进行比较。
- 每一个比较器都可以进行配置，以监测 CPU 或 XGATE 总线。
- 每一个比较器都对读/写和字节/字访问周期进行控制。
- 比较结果可以作为状态序列发生器的触发条件。
- 3 种比较器模式：
  - 简单的地址/数据比较器匹配模式。
  - 地址范围内模式，最小地址 ≤ 地址 ≤ 最大地址。
  - 地址范围外匹配模式，地址 < 最小地址或地址 > 最大地址。
- 2 种触发器类型：
  - 标记的——在特定的指令执行之前触发。
  - 强迫的——在匹配发生后、第一条指令之前触发。
- 3 种类型的断点：
  - 断点后 CPU 进入 BDM。
  - 断点后 CPU 执行 SWI。
  - XGATE 断点。
- 与比较器无关的 3 种触发模式：
  - 外部指令标记（只与 CPU 指令相关）。
  - XGATE S/W 断点请求。
  - TRIG 为立即软件触发器。
- 3 种跟踪模式：
  - 常规——保存流变化（change of flow, COF）总线信息，用于定义流变化。
  - Loop1——与常规模式相同，但是对连续重复源地址的进入进行一致。
  - 详细——保存除自由周期和取操作码之外的所有周期的地址和数据。
- 4 站的状态序列发生器用于跟踪缓冲区控制：
  - 跟踪与状态序列发生器的最终状态相连接的序列触发器。
  - 触发器跟踪的开始、结束和中间对齐。

在所有 MCU 功能模式中都可以使用 DBG 模块。在 BDM 硬件访问中，以及当 BDM 模块处于活动状态时，CPU 监视被禁止。因此，映射到 CPU 的断点、比较器和总线追踪也被禁止；但是对 DBG 寄存器的访问仍然可以进行，包括比较寄存器。当 BDM 模块处于活动状态时，或者在硬件 BDM 访问中，XGATE 的行为仍然可以进行比较、跟踪，并可以用于向 XGATE 模块产生一个断点。当通过 BACKGROUND 命令使 CPU 进入 BDM 有效模式时，DBG 模块开始

运行。如果 MCU 处于保护模式，DBM 模块跟踪被禁止。然而，在 MCU 保护模式中仍然可以产生断点。

### 12.1.16 外部总线接口

外部总线接口（External Bus Interface, XEBI）控制着与芯片操作模式相关的非多路外部总线（“扩充总线”）。根据所处的模式，外部总线可以被用于外部存储器、外设或者 PRU 之间的数据交换；在模拟器的帮助下，还可以从外部查看内部总线（见图 12-33）。

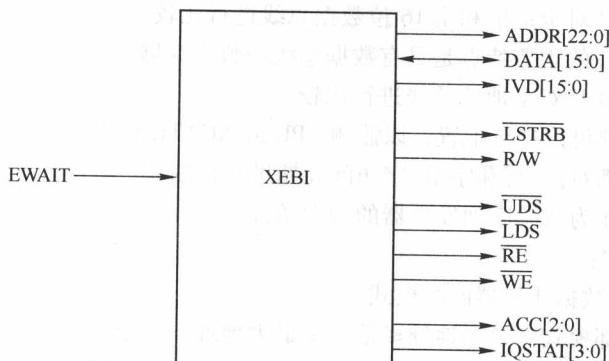


图 12-33 XEBI 框图 (Freescale 2007 版权许可)

### 特点

XEBI 具有如下特点：

- 可以输出最多 23 位的地址总线和控制信号，可以与非多路外部总线一起使用。
- 双向的 16 位外部数据总线，可以选择将高 8 位禁止。
- 可以观察内部总线行为。

### 12.1.17 端口综合模块

S12XD 系列端口综合模块（Port Integration Module, PIM）在外设模块之间建立了接口，包括非多路外部总线接口模块（S12\_EBI）和所有端口的 I/O 引脚（见图 12-34）。它可以控制电气引脚特性和信号优先级，并对共享的引脚进行多路控制。

PIM 有如下端口：

- 端口 A 和 B 用作 S12X\_EBI 的地址输出。
- 端口 C 和 D 用作 S12X\_EBI 的数据 I/O。
- 端口 E 与 S12X\_EBI 控制信号和 IRQ、XIRQ 中断输入有关。
- 端口 K 与 S12X\_EBI 的地址输出和控制信号有关。
- 端口 T 与增强型捕捉定时器（ECT）模块连接。
- 端口 S 与 2 个 SCI 和 1 个 SPI 模块相关。
- 端口 M 与 4 个 MSCAN 模块和 1 个 SCI 模块相关。
- 端口 P 与 PWM 和 2 个 SPI 模块相连，输入可以作为外部中断源。
- 端口 H 与 2 个 SCI 模块相关，输入可以作为外部中断源。
- 端口 J 与 1 个 MSCAN、1 个 SCI 和 2 个 IIC 端口相关，输入可以作为外部中断源 54ff。
- 端口 AD0 和 AD1 与一个 8 通道和一个 16 通道 ATD 模块相关。

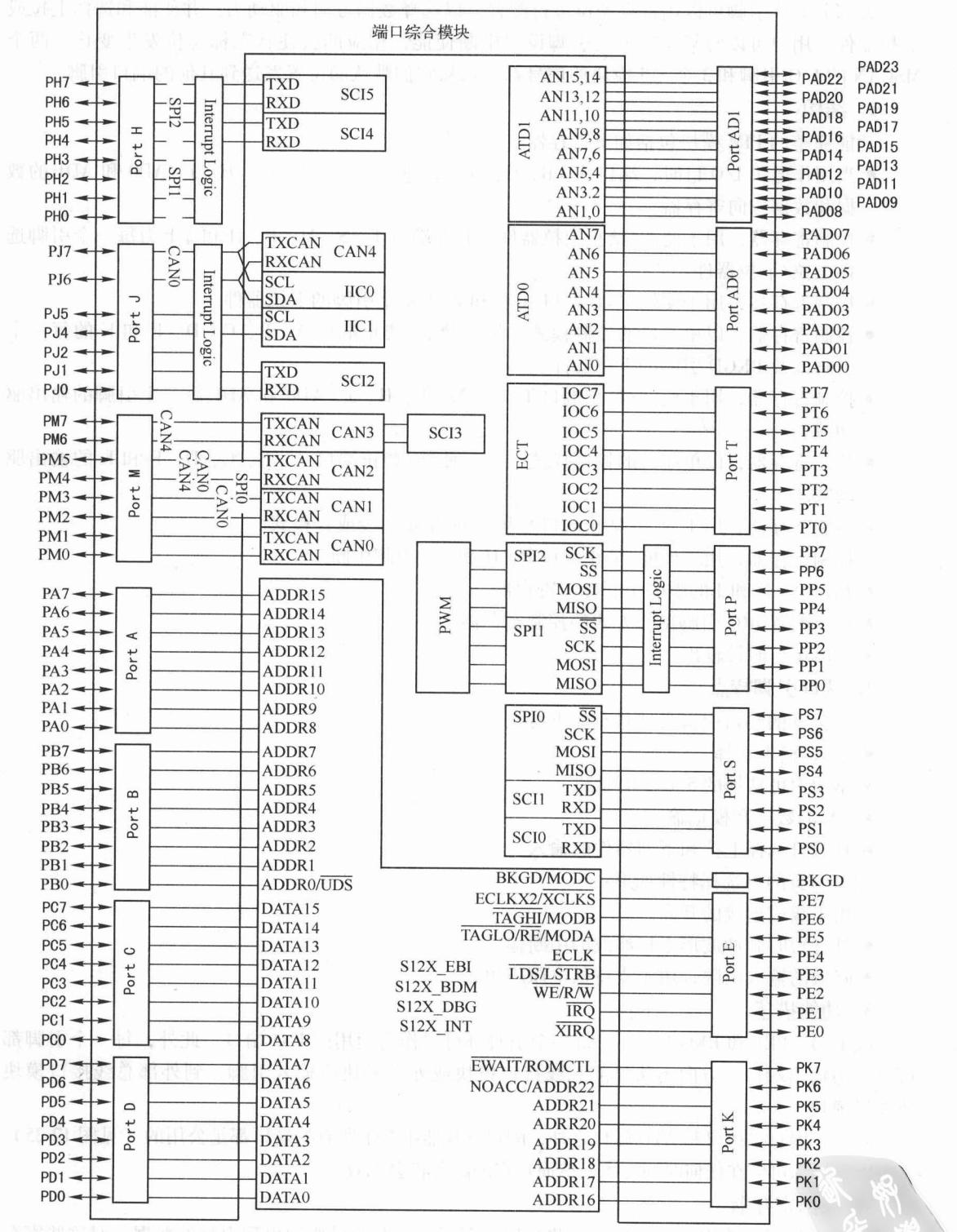


图 12-34 PIM 框图 (Freescale 2007 版权许可)

大部分 I/O 引脚可以用寄存器位进行配置，以选择数据方向和驱动力，并使能和选择上拉或下拉器件。用户可以对某些特定的引脚设置中断使能，相应的会使状态标志位发生变化。两个 MSCAN 的 I/O 引脚和 3 个 SPI 模块的信号都可以从它们默认的位置发送到其他的端口引脚。

## 1. 结构

功能完整的 PIM 模块包括如下寄存器：

- 当用作通用 I/O 口时，端口 A、B、C、D、E、K、T、S、M、P、H、J、AD0 和 AD1 的数据和数据方向寄存器。
- 控制寄存器，用于使能/禁止上拉器件，并在端口 T、S、M、P、H 和 J 上为每一个引脚选择上拉/下拉器件。
- 控制寄存器，用于使能/禁止端口 AD0 和 AD1 每个引脚的上拉器件。
- 控制寄存器，以单端口的控制模式，用于使能/禁止端口 A、B、C、D、E 和 K 的每一个引脚以及 BKGD 引脚的上拉器件。
- 控制寄存器，用于使能/禁止端口 T、S、M、P、H、J、AD0 和 AD1 每一个引脚的输出驱动减弱。
- 控制寄存器，以单端口的控制模式，用于使能/禁止端口 A、B、C、D、E 和 K 的输出驱动减弱。
- 控制寄存器，用于使能/禁止端口 S 和 M 的开漏（线或）模式。
- 控制寄存器，用于使能/禁止端口 P、H 和 J 的引脚中断。
- 端口 P、H 和 J 的引脚中断标志寄存器。
- 用于配置 IRQ 引脚工作方式的控制寄存器。
- 自由运行时钟输出。

## 2. 端口引脚特点

一个标准的端口引脚必须具有如下特点：

- 输入/输出选择。
- 驱动力可选择的 5 V 输出驱动。
- 5 V 的数字和模拟输入。
- 具有可选择上拉和下列器件的输入。

端口引脚的可选择特性如下：

- 用于线或连接的开漏。
- 具有短时脉冲波形干扰滤波的中断输入。
- 降低的输入门限，用于支持低电压应用。

## 3. 功能描述

除 PE0、PE1 和 BKGD 之外，每一个引脚都可以作为通用的 I/O 端口。此外，每一个引脚都可以作为输出端口，方向为从外部总线接口模块或外设模块或输入引脚，到外部总线接口模块或外设模块。

除了扩展总线接口和 ATD 端口之外，配置寄存器组对于所有的端口都是公用的（见图 12-35）。所有寄存器都可以在任何时间写入，然而特定的配置将会无效。

## 4. 数据寄存器

如果引脚用作通用 I/O 端口，这些数据寄存器中保存的是要输出到引脚的数据。对这些寄存器进行写入操作，只能影响被设置为通用输出端口的引脚。当对这些地址进行读操作时，如果相应的数据方向寄存器位为 0，则会返回引脚的缓冲状态。

端口	数据	数据方向	输入	驱动减弱	上拉使能	极性选择	线或模式	中断使能	中断标志
A	是	是	—	是	是	—	—	—	—
B	是	是	—			—	—	—	—
C	是	是	—			—	—	—	—
D	是	是	—			—	—	—	—
E	是	是	—			—	—	—	—
K	是	是	—			—	—	—	—
T	是	是	是	是	是	—	—	—	—
S	是	是	是	是	是	是	—	—	—
M	是	是	是	是	是	是	—	—	—
P	是	是	是	是	是	是	—	是	是
H	是	是	是	是	是	是	—	是	是
J	是	是	是	是	是	是	—	是	是
AD0	是	是	—	是	是	—	—	—	—
AD1	是	是	—	是	是	—	—	—	—

每一个单元代表一个单独配置位的寄存器。

图 12-35 每个端口所用的寄存器 (Freescale 2007 版权许可)

如果数据方向寄存器位被设置为高电平 1，则会返回数据寄存器的值。这一点与图 12-36 所示的任何其他配置都是无关的。

## 5. 输入寄存器

输入寄存器是只读寄存器，当读取时，总是返回引脚的缓冲状态，如图 12-36 所示。

## 6. 数据方向寄存器

该寄存器的值定义了引脚是用作输入端口，还是输出端口。如果引脚是由外部模块控制的，数据方向寄存器的值将不起作用（见图 12-36）。

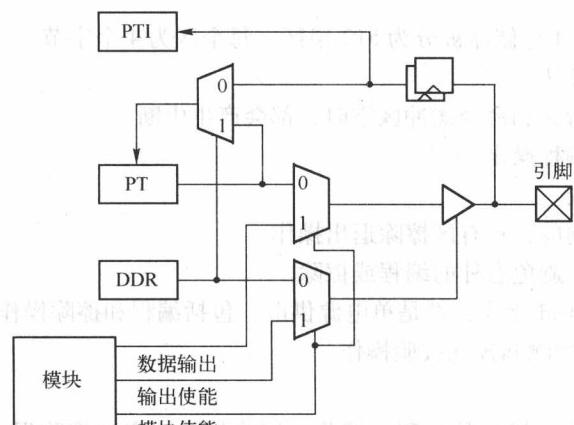


图 12-36 I/O 引脚功能描述 (Freescale 2007 版权许可)

### 12.1.18 2K 字节 EEPROM (EETX2K)

EETX2K 模块是 2K 字节 EEPROM (非易失性) 存储器块 (见图 12-37)，除此外还有一个 4K 字节的 EEPROM 块。对 EEPROM 存储器的读取单位可以是字节、对齐字或者非对齐的字。对于按字节和对齐字访问的存储器，读访问时间是 1 个总线周期；而对于非对齐字的访问，读访问

时间是 2 个总线周期。

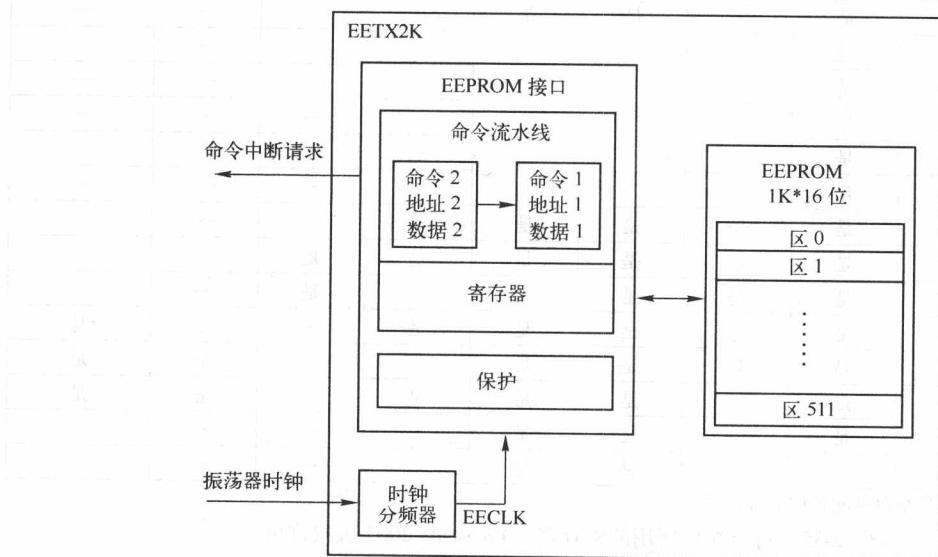


图 12-37 EEPROM 框图 (Freescale 2007 版权许可)

EEPROM 存储器非常适用于单通道应用的数据存储，因为它允许现场重编程，对于编程和擦除操作，都不需要外部电源。编程和擦除功能由命令驱动接口来控制。EEPROM 模块不仅支持块擦除（所有的存储器字节），还支持区擦除（4 个存储器字节），被擦除的位为 1，被编程的位为 0。编程和擦除 EEPROM 存储器的高电压是由内部产生的。当对 EEPROM 块进行擦除或编程时，不能同时进行读操作。

## 1. 特点

- 2K 字节的 EEPROM 存储器被分为 512 扇区，每个区为 4 个字节。
  - 自动编程和擦除算法。
  - EEPROM 命令执行完和命令缓冲区空时，都会产生中断。
  - 快速区擦除和字编程操作。
  - 两站的命令流水线。
  - 对于关键的中断响应，具有区擦除退出操作。
  - 灵活的保护方案，避免意外的编程或擦除。
  - 对于所有的 EEPROM 操作，都是单电源供电，包括编程和擦除操作。
- EEPROM 支持编程、擦除和擦除校验操作。

## 2. 功能描述

写操作用于执行本节中描述的编程、擦除、擦除校验、扇区擦除退出和扇区修改算法。编程、擦除和扇区修改算法是由一个状态机控制的，该状态机的基准时钟是 EECLK，是通过可编程的分频器从振荡器时钟获得的。

命令寄存器和相关的地址和数据寄存器（见图 12-38）用作一个缓冲区和一个寄存器（两站的 FIFO），这样在第一条命令正在执行的同时，第二个命令以及必要的数据和地址可以保存在缓冲区中。缓冲区空和命令结束都会使 EEPROM 状态寄存器发生变化，如果中断被设置为使能，还可以发出中断。

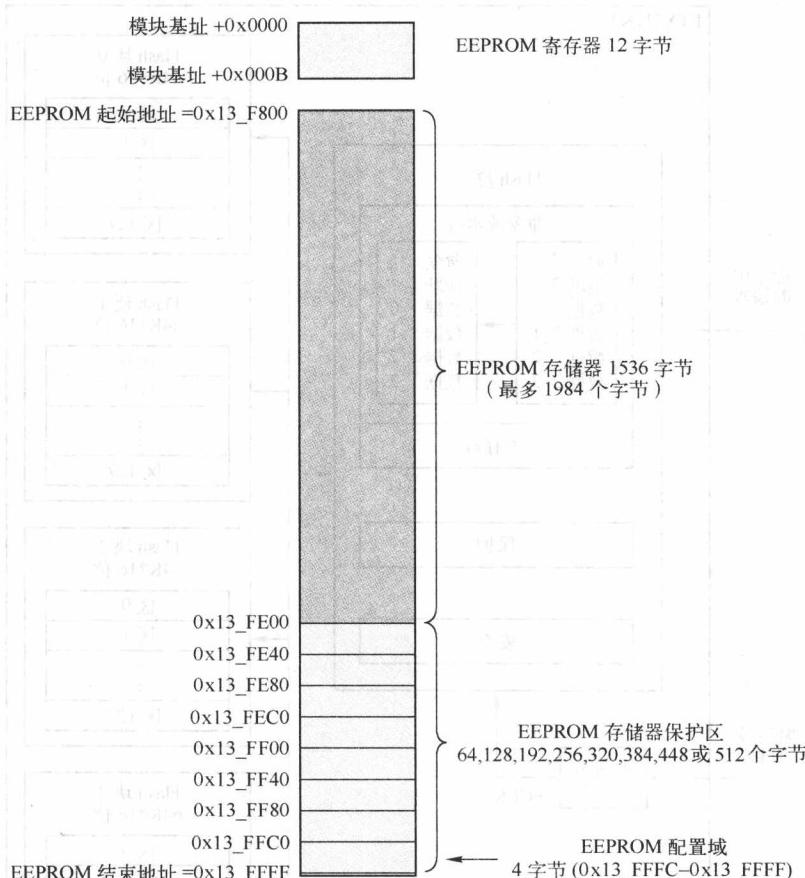


图 12-38 2K 字节 EEPROM 存储器映射 (Freescale 2007 版权许可)

### 3. EEPROM 模块安全

EEPROM 模块不向 MCU 提供任何安全信息。在每次重新启动之后，MCU 的安全状态都由 Flash 模块信息来决定。

#### 12.1.19 512K 字节 Flash 模块 (FTX512K4)

EEPROM 存储器非常适用于单通道应用的程序和数据存储，因为它允许现场重编程，对于编程和擦除操作，都不需要外部电源（见图 12-39）。编程和擦除功能由命令驱动接口来控制。Flash 模块不仅支持块擦除，还支持扇区擦除。被擦除的位为 1，被编程的位为 0。编程和擦除 Flash 存储器的高电压是由内部产生的。当对 Flash 块进行擦除或编程时，不能同时进行读操作。

#### 特点

FTX512K4 具有如下特点：

- 512K 字节的 EEPROM 存储器被分为 4 个 128 字节的块，每一个块被分为 128 个区，每个区为 1024 个字节。
- 自动编程和擦除算法。
- flash 命令执行完和命令缓冲区空时，都会产生中断。
- 快速区擦除和字编程操作。
- 两站的命令流水线，用于提高多字编程速度。

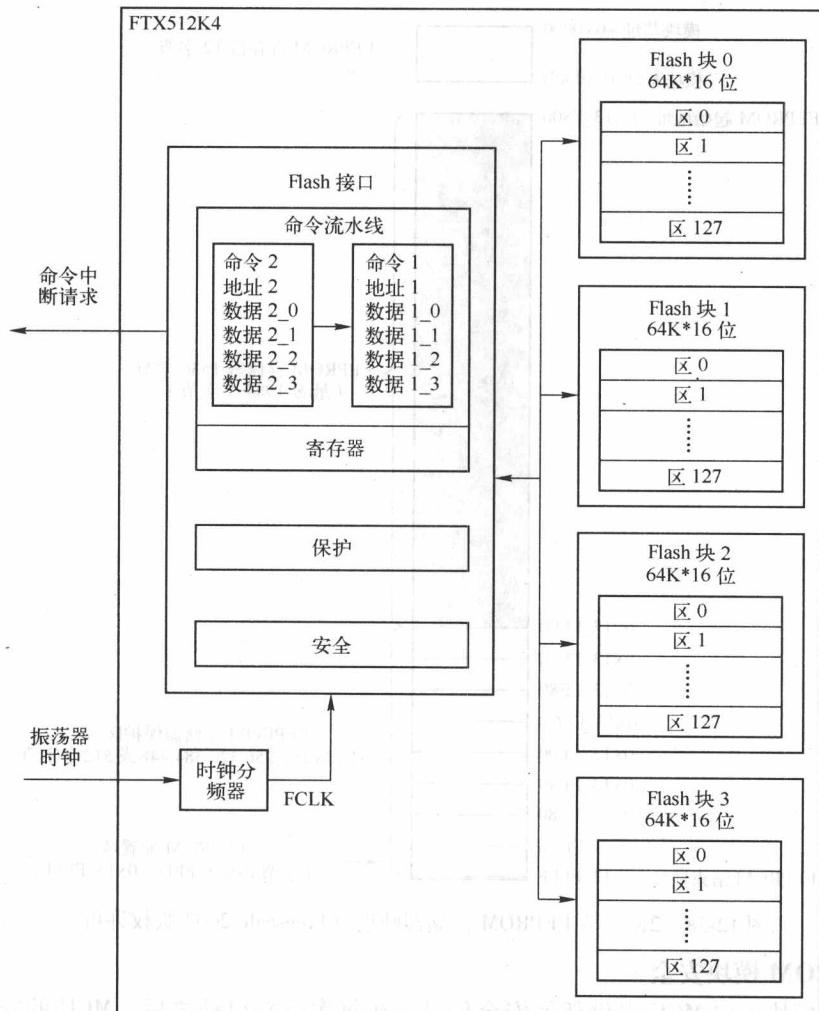


图 12-39 Flash 框图 (Freescale 2007 版权许可)

- 对于关键的中断响应，具有扇区擦除退出操作。
- 对于所有的 Flash 操作，都是单电源供电，包括编程和擦除操作。
- 灵活的保护方案，避免对 Flash 存储器进行未经授权的访问。
- 用内建的压缩数据，进行代码完整性检查。

512K 字节的 Flash 存储器可以以字节、对齐字或非对齐字为单位进行读取。对于按字节和对齐字访问的存储器，读访问时间是 1 个总线周期；而对于非对齐字的访问，读访问时间是 2 个总线周期。

### 12.1.20 安全性

该模块提供基于硬件的安全措施。必须记住一点，安全性 (Security, SEC) 的一部分与应用代码有关。作为一个极端的例子，例如有一段应用程序代码将内部存储器的内容倒出。这样就违背了安全性的目的。同时，在应用程序代码中，最好要留一个后门。例如通过 SCI 下载安全密钥，这个操作将会访问编程子程序，修改保存在 Flash 存储器另外一个区的更新参数。

#### 1. 特点

S12X 系列芯片的安全特性（在安全模式下）如下。

- 保护非易失性存储器的内容 (Flash、EEPROM)。
- 限制 NVM 命令的执行。
- 禁止通过背景调试模块 (BDM) 访问内部存储器。
- 禁止在扩展模式中访问内部 Flash/EEPROM。
- 禁止 CPU 和 XGATE 的调试特征。

安全的操作对于微控制器的影响见图 12-40。

	非安全模式						安全模式					
	NS	SS	NX	ES	EX	ST	NS	SS	NX	ES	EX	ST
Flash 存储器访问	✓	✓	✓ <sup>1</sup>	✓ <sup>1</sup>	✓ <sup>1</sup>	✓ <sup>1</sup>	✓	✓	—	—	—	—
EEPROM 存储器访问	✓	✓	✓	✓	✓	✓	✓	✓	—	—	—	—
NVM 命令	✓ <sup>2</sup>	✓	✓ <sup>2</sup>	✓ <sup>2</sup>	✓ <sup>2</sup>	✓	✓ <sup>2</sup>					
BDM	✓	✓	✓	✓	✓	✓	—	✓ <sup>3</sup>	—	—	—	—
DBG 模块追踪	✓	✓	✓	✓	✓	✓	—	—	—	—	—	—
XGATE 调试	✓	✓	✓	✓	✓	✓	—	—	—	—	—	—
外部总线接口	—	—	✓	✓	✓	✓	—	—	✓	✓	✓	✓
内部状态在外部总线上通过多路开关可见	—	—	—	✓	✓	—	—	—	—	✓	✓	—
内部访问在外部总线上可见	—	—	—	—	—	✓	—	—	—	—	—	✓

1. 存储器映射中 Flash 阵列的可用性，依赖于 ROMCTL/EROMCTL 引脚和 / 或 MMCCTL1 寄存器中 ROMON/EROMON 位的状态。对于详细的信息，请读者参考 S12X\_MMC 块指南。

2. 仅限于 NVM 命令集。对于详细的信息，请参考 FTX/EETX 块指南。

3. BDM 硬件命令只限于外设寄存器。

图 12-40 在非安全和安全模式下的特性 (Freescale 2007 版权许可)

## 2. 操作模式

当 Flash 和 EEPROM 被编程后，通过对芯片安全位编程进行保护，该安全位位于 Flash 存储器阵列中的选项/安全字节中。这些非易失的信息可以在设备运行期间保持安全性。

## 3. 保护的微控制器

通过保护设备，可以避免对 EEPROM 和 Flash 存储器未经授权的访问。然而必须知道，EEPROM 和 Flash 存储器内容的安全性还取决于应用程序的设计。例如，如果应用程序能够通过串行总线下载并执行代码（即应用程序中包含引导代码），那么即使微控制器处于安全状态中，这个功能也可能会用于读取 EEPROM 和 Flash 存储器的内容。在上述的例子中，通过在代码下载之前增加一个挑战/应答认证机制，可以提高应用程序的安全性。

## 12.2 Texas Instruments MSP430™ 系列

MSP430™16 位 RISC CPU，使用冯·诺依曼公用存储器地址总线 (MAB) 和存储器数据总线 (MDB)，将外设和灵活的时钟系统相结合，如图 12-41 所示。该处理器采用类 RISC 的指令集，大部分指令可以在一个周期内执行完。MSP430 系列处理器中引入了一种现代 CPU，具有标准组建存储器映射的模拟和数字外设，对于混合信号应用给出了很好的解决方案。

Texas Instruments MSP430 系列超低功耗微控制器中包含了许多设备，针对不同的应用，可以设置不同的外设组合。该体系结构与 5 种低功耗模式相结合，非常适合于便携式产品，可以延长电池的寿命。控制器的特点是，具有强大的 16 位 RISC CPU、16 位寄存器和常数产生器，可以最大限度地提高代码效率。

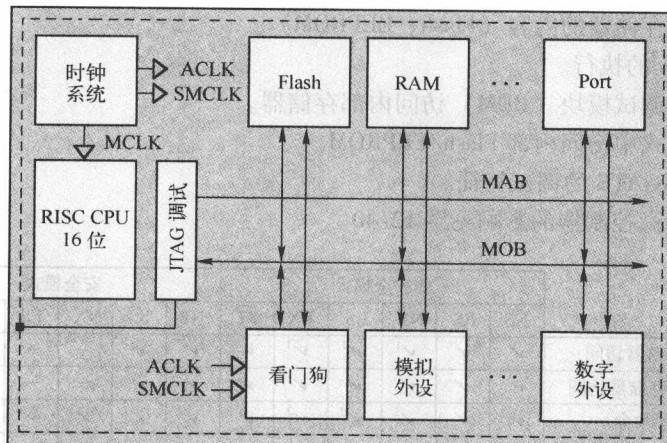


图 12-41 MSP430 体系结构 (经德州仪器许可使用)

MSP430 MCU 的正交体系结构具有很好的灵活性，提供 16 个全地址可寻址的、单周期 16 位 CPU 寄存器（在 MSP430 体系结构中是 20 位）和 RISC 功能（见图 12-42）。现代 CPU 的设计中仅使用 27 条易于理解的指令和 7 种相容寻址模式来提供多功能性，这样的设计可以形成 16 位的低功耗 CPU，处理能力更高，体积更小，执行代码的效率更高。

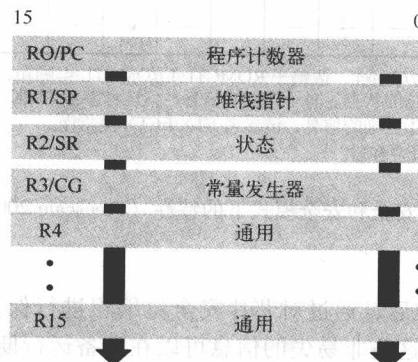


图 12-42 MSP430 寄存器组 (经德州仪器许可使用)

MSP430 系列控制器框图如图 12-43 所示，其基本的配置如下：

- 1KB ~ 120KB 的 ISP Flash。
- 最多 10KB 的 RAM。
- 14 ~ 100 个引脚的选择。

MSP430 系列控制器框图如图 12-43 所示，其超低功耗特性如下：

- 多种操作模式。
- 0.1 $\mu$ A 的节电模式功耗。
- 0.8 $\mu$ A 的待机模式功耗。
- 250 $\mu$ A/MIPS@ 3V。
- 即时启动稳定的高速时钟。
- 1.8 ~ 3.6V 的工作电压。
- 零功耗 BOR。
- <50nA 引脚漏电流。
- 每项任务具有最小的 CPU 时钟数。

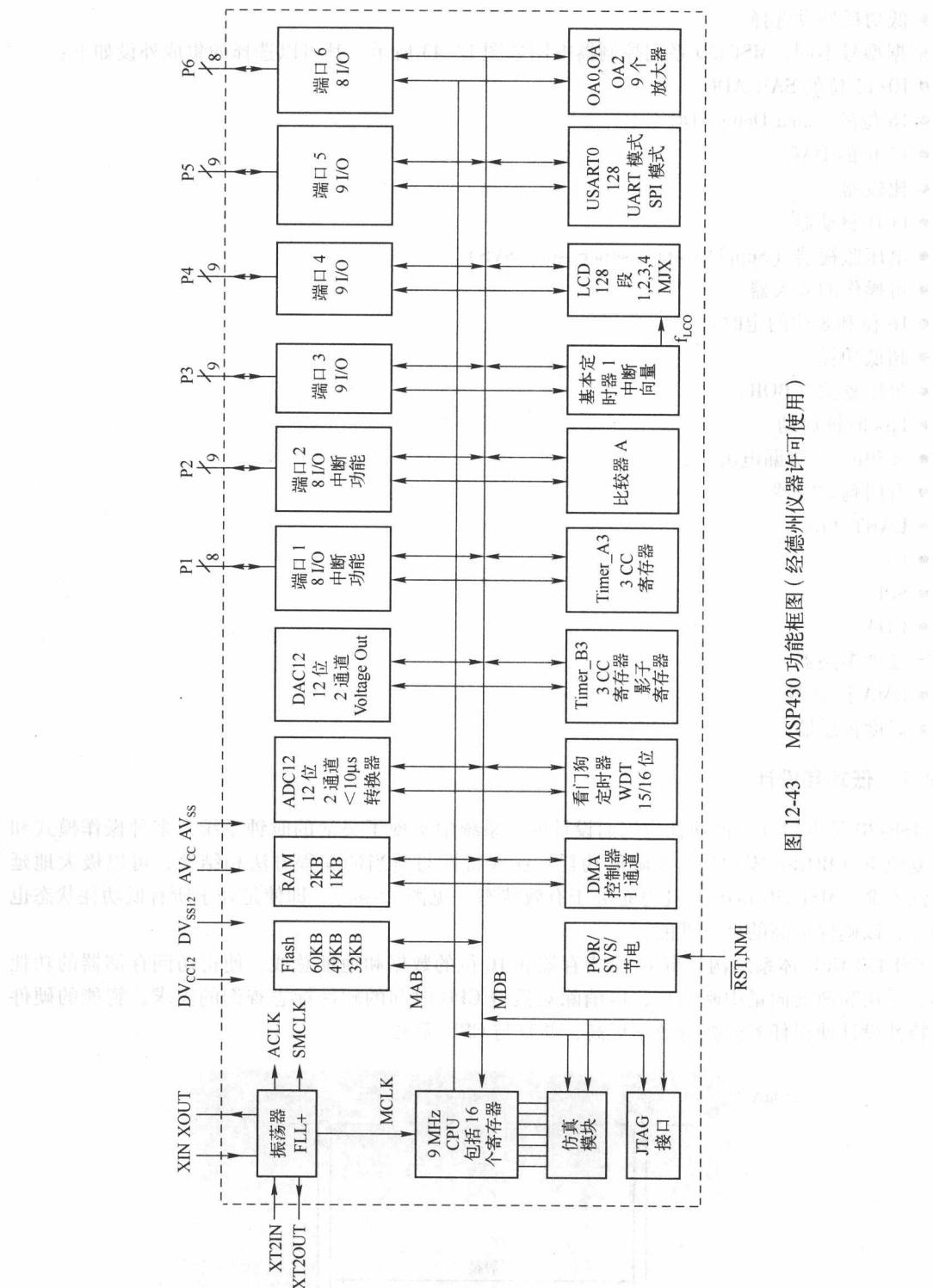


图 12-43 MSP430 功能框图(经德州仪器许可使用)

- 低功耗外设选择。

根据型号不同, MSP430 系列控制器框图如图 12-43 所示, 其可以选择的集成外设如下:

- 10/12 位的 SAR ADC。
- 16 位的 Sigma Delta ADC。
- 12 位的 DAC。
- 比较器。
- LCD 驱动器。
- 电压监视器 (Supply voltage supervisor, SVS)。
- 可操作的放大器。
- 16 位和 8 位的定时器。
- 超低功耗。
- 欠压复位 (BOR)。
- $1\mu\text{s}$  时钟启动。
- $<50\text{nA}$  引脚漏电流。
- 看门狗定时器。
- UART/LIN。
- I<sup>2</sup>C。
- SPI。
- IrDA。
- 硬件多路器。
- DMA 控制器。
- 温度传感器。

### 12.2.1 低功耗设计

MSP430 是专门针对低功耗应用而设计的, 系统中实现了灵活的时钟系统、多种操作模式和欠压复位下 (BOR) 零功耗, 以降低功耗。这些特性与适当的编程方法相结合, 可以极大地延长电池寿命。MSP430 BOR 功耗总是处于有效状态 (见图 12-44), 即使是处于所有低功耗状态也不例外, 以确保最高的可靠性能。

MSP430 CPU 体系结构具有 16 个寄存器和 16 位的数据和地址总线, 使得访问存储器的功耗最小; 采用高速的向量中断结构, 以消除对浪费 CPU 时间的软件标志查询的需求; 智能的硬件外设特性设计使得任务的执行效率更高, 并且与 CPU 无关。

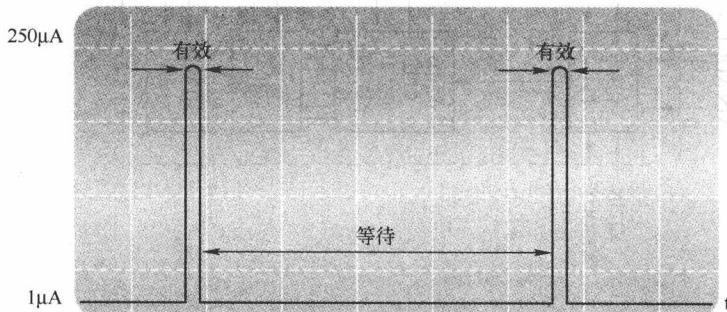


图 12-44 MSP430 功耗状况 (经德州仪器许可使用)

## 12.2.2 灵活的时钟系统

MSP430 CPU 的时钟系统是专门针对电池供电的应用而设计的，其中使用了多个振荡器（见图 12-45），以支持事件驱动的突发行为。低频辅助时钟（A low-frequency Auxiliary Clock，ACLK）直接由公用的 32 kHz 晶振或内部超低功耗振荡器（very low power oscillator，VLO）驱动，不需要其他外部部件。ACLK 可以用于背景实时时钟字唤醒功能。

集成的高速数控振荡器（digitally controlled oscillator，DCO）可以为 CPU 使用的主设备时钟（master clock，MCLK）和高速外设使用的二级时钟（submain clock，SMCLK）提供振荡源。根据设计，DCO 在 9  $\mu$ s（F2XX）或者 6  $\mu$ s（X1XX，X4XX）时处于有效和稳定状态，这样可以实现高性能和超低功耗的需求。

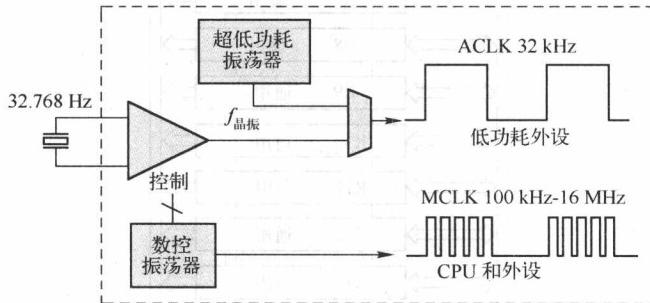


图 12-45 多振荡器时钟系统（经德州仪器许可使用）

## 12.2.3 MSP430 CPU

MSP430 CPU 具有 16 位的 RISC 体系结构，并且对应用是高度透明的。除程序流指令之外，所有操作都是寄存器操作；并且对源操作数采用了 7 种寻址方式，对于目的操作数采用了 4 种寻址方式。

MSP430 CPU 中集成了 16 个 16 位的寄存器，可以降低指令的执行时间。MSP430 CPU 中有 20 位寄存器，如图 12-46 所示。寄存器到寄存器操作的执行时间是 1 个 CPU 时钟周期。其中有 4 个寄存器——R0 ~ R3，只能分别作为程序计数器、堆栈指针、状态寄存器和常数发生器。其余寄存器都是通用寄存器。

外设通过数据、地址和控制总线与 CPU 相连，可以由所有指令进行操作。指令集共包括 51 条指令（27 条内核指令和 24 条仿真指令），具有 3 种格式和 7 种寻址方式，每一条指令都对字和字节数据进行处理。内核指令指的是由 CPU 译码、具有唯一操作码的指令。

仿真指令指的是提高代码的可读性和可写性但本身并不具有唯一操作码的指令。在编译时，这些指令可以由编译器自动替换为等价的内核指令。使用仿真指令，并没有代码和性能方面的代价。

内核指令格式共有如下 3 种：

- 双操作数。
- 单操作数。
- 跳转。

所有单操作数和双操作数指令都可以是字节或字指令，字节指令用于访问字节数据或者字节外设，字指令用于访问字数据或字外设。

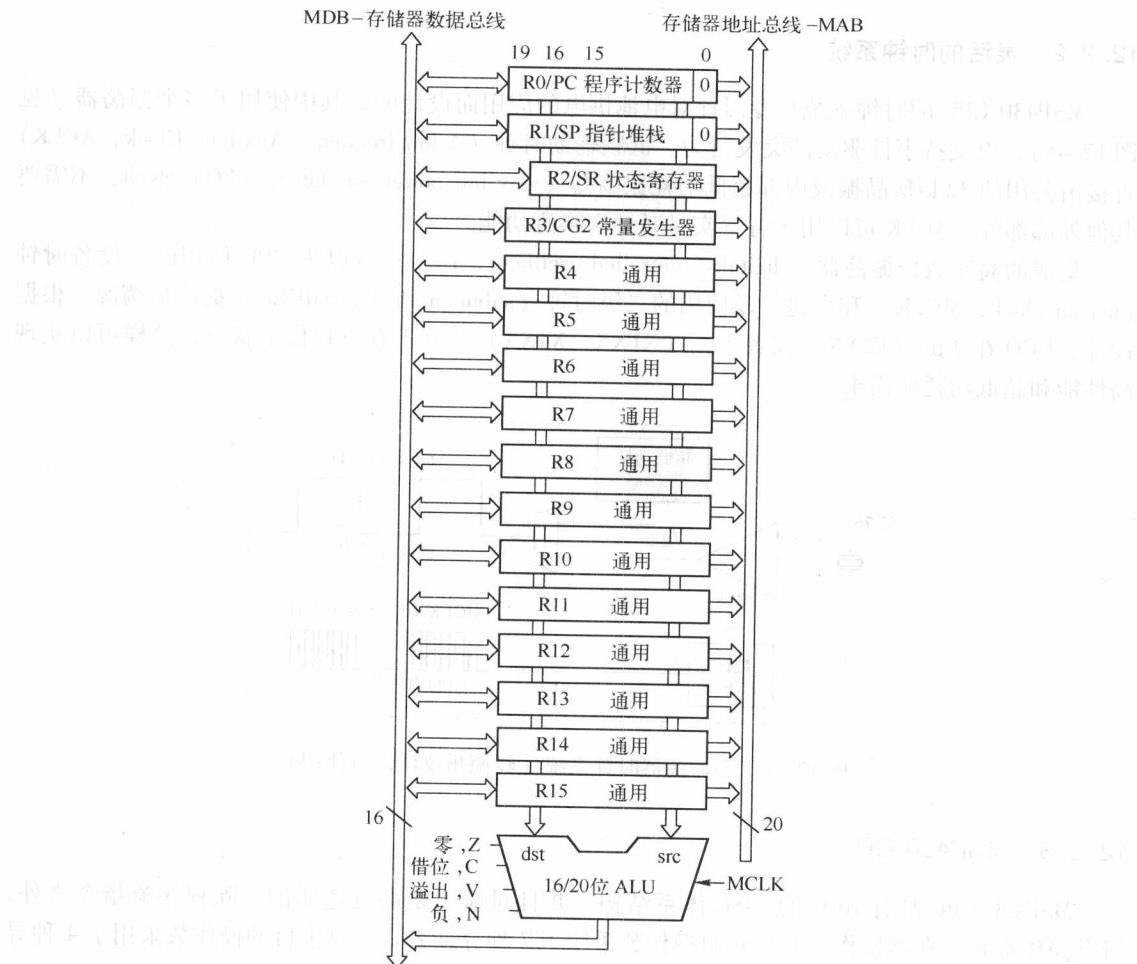


图 12-46 MSP430 CPU 框图（经德州仪器许可使用）

### 12.2.4 操作模式

MSP430 具有 1 种主动模式和 5 种软件可选择的低功耗操作模式。中断事件可以将设备从 5 种低功耗模式中的任何一种唤醒，对中断请求进行服务；并在从中断程序中返回时，恢复到低功耗状态。

如下 6 种操作模式可以由软件进行配置：

- 有效模式 (Active mode, AM)：
  - ◆ 所有时钟都处于有效状态。
- 低功耗模式 0 (Low-power mode 0, LPM0)：
  - ◆ CPU 被禁止；
  - ACLK 和 SMCLK 处于有效状态，MCLK 被禁止；
  - FLL + loop 控制处于有效状态。
- 低功耗模式 1 (LPM1)。
  - ◆ CPU 被禁止；
  - FLL + loop 控制被禁止；
  - ACLK 和 SMCLK 处于有效状态，MCLK 被禁止。
- 低功耗模式 2 (LPM2)：

- CPU 被禁止;
  - MCLK、FLL + loop 控制和 DCOCLK 被禁止;
  - DCO 的 dc - 发生器处于使能状态;
  - ACLK 处于有效状态。
- 低功耗模式3 (LPM3):
- CPU 被禁止;
  - MCLK、FLL + loop 控制和 DCOCLK 被禁止;
  - DCO 的 dc - 发生器被禁止;
  - ACLK 处于有效状态。
- 低功耗模式4 (LPM4):
- CPU 被禁止;
  - ACLK 被禁止;
  - MCLK、FLL + loop 控制和 DCOCLK 被禁止;
  - DCO 的 dc - 发生器被禁止;
  - 晶振被停止。

### 12.2.5 FLL<sup>+</sup>时钟模块

FLL<sup>+</sup> (The frequency-locked loop, 锁频环) 时钟模块支持低成本和超低功耗, 框图如图 12-47 所示。该模块共使用了 3 个内部时钟信号, 可以对性能和低功耗进行最佳平衡。

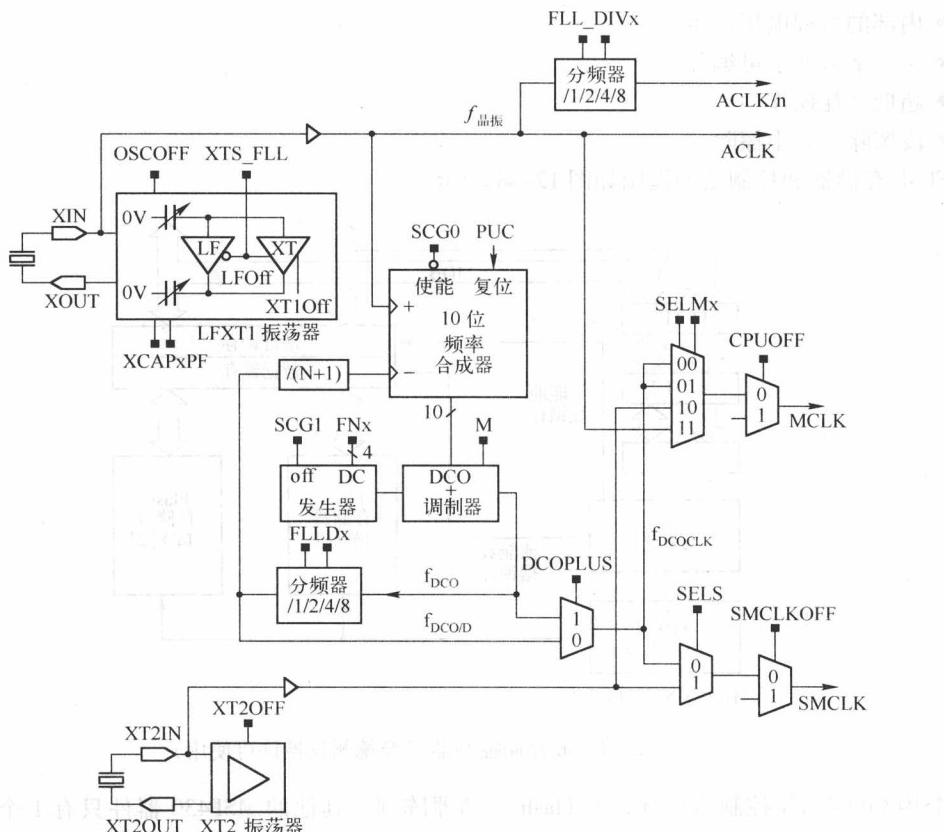


图 12-47 MSP430 锁频环 (经德州仪器许可使用)

$\text{FLL}^+$ 的特点是具有锁频环（FLL）硬件。FLL与数字调节器一起工作，将内部数控制振荡器（DCO）频率稳定为可编程的LFXT1晶振频率的倍数；可以对FLL<sup>+</sup>时钟模块进行配置，使其不需要任何外部部件，而只需要一个或两个外部晶振，或者软件控制的振荡器。

在电池供电的 MSP430 应用中，通常会存在如下典型的需求冲突。

- 为了节省功耗降低时钟频率和时间同步之间的冲突。
  - 为了对时间的快速响应需要高的时钟频率和快速突发处理能力之间的冲突。
  - 工作温度下的时钟稳定性和供电电压之间的冲突。

FLL<sup>+</sup>时钟模块通过对 ACLK、MCLK 和 SMCLK 3 种时钟信号进行选择，处理上述的冲突需求。为了获得最佳的低功耗性能，可以将 ACLK 配置为使用一种低功耗的 32 786 Hz 晶振，位系统和低功耗暂停操作提供稳定的时间基准；可以将 MCLK 配置为使用片上 DCO 进行工作，DCO 由 FLL 稳定获得，并且当有中断事件需求时，可以激活。

与锁相环相比，数字锁频环可以降低启动时间并稳定延迟。锁相环需要几百个或几千个时钟周期进行启动和稳定，而根据先前的设置，FLL 可以立即启动。

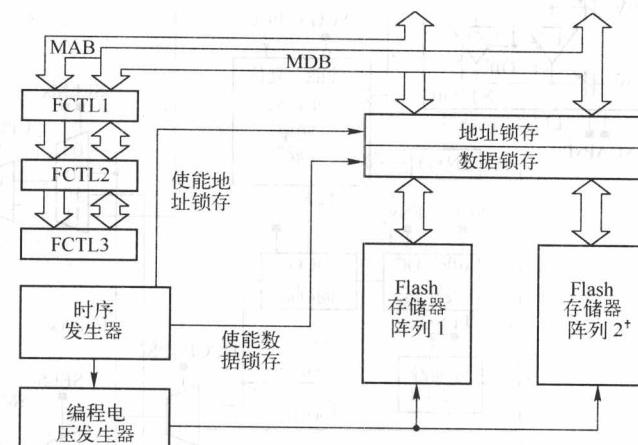
### 12.2.6 Flash 存储控制器

MSP430 Flash 存储器是位、字节和字可寻址和可编程的。Flash 存储器模块具有一个集成的控制器，控制着编程和擦除操作。控制器具有 3 个或 4 个寄存器（参见特定器件的数据手册）、1 个定时发生器和 1 个电压发生器，用于给编程和擦除操作提供电压。

MSP430 Flash 存储器的特点如下：

- 内部的编程电压产生。
  - 位、字节或字可编程。
  - 超低功耗操作。
  - 段擦除和集中擦除。

Flash 存储器和控制器的框图如图 12-48 所示。



<sup>†</sup>只针对 MSP430FG461x。

图 12-48 Flash 存储控制器（经德州仪器许可使用）

MSP430FG461X 控制器具有 2 个 Flash 存储器阵列，其他的 MSP430 器件只有 1 个存储器阵列。所有 Flash 存储器被分成段，可以向 Flash 存储器中写入一个位、字节或字，但是段是 Flash

存储器可以擦除的最小单位。

Flash 存储器被分为主存储区和信息存储器区，对主存储区和信息存储器区的操作没有什么区别，代码和数据可以位于任何一个区。两个区之间的区别在于段尺寸和物理地址。对于 F47x 器件，信息存储器具有 4 个 64 字节的段；对于其他的所有 4xx 器件，具有 2 个 128 字节的段。主存储器具有 2 个或更多 512 字节的段，段会再进一步被划分为块。图 12-49 以 4K 字节的 Flash 存储器为例介绍了 Flash 的分段情况，共有 8 个主存储段和 2 个信息存储段。

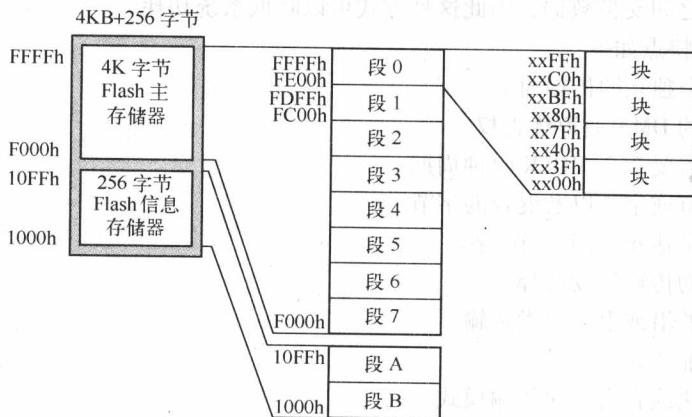


图 12-49 Flash 存储器段 (4KB) (经德州仪器许可使用)

### 12.2.7 硬件多路器

硬件多路器是一个外设，并不是 MSP430 CPU 的一部分，也就是说它的行为并不影响 CPU 的行为。多路器寄存器是外设寄存器，由 CPU 指令进行写入和读取操作。

硬件乘法器支持无符号乘法、有符号乘法、无符号乘加、有符号乘加、 $16 \times 16$  位、 $16 \times 8$  位、 $8 \times 16$  位和  $8 \times 8$  位操作，操作的类型由第一操作数写入的地址进行选择。硬件乘法器框图如图 12-50 所示。

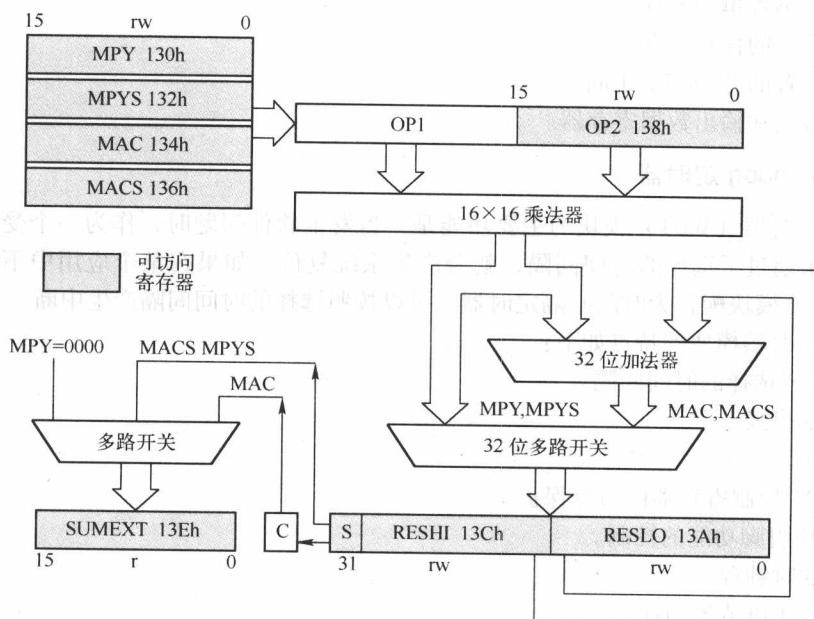


图 12-50 硬件乘法器 (经德州仪器许可使用)

### 12.2.8 DMA 控制器

DMA (direct memory access, 直接存储器存取) 控制器可以在不需要 CPU 干预的情况下，在全地址范围内，将数据从一个地址传输到另外一个地址。例如，DMA 控制器可以将数据从 ADC12 变换存储器传输到 RAM 中。包含 DMA 控制器的处理器可以具有 1 个、2 个或 3 个 DMA 通道，使用 DMA 控制器可以提高外设模块的吞吐量。由于可以在 CPU 保持低功耗状态下，在外设和系统之间交换数据，因此这种方式可以降低系统功耗。

DMA 控制器的特点如下：

- 具有多达 3 个独立的传输通道。
- 具有可配置的 DMA 通道优先权。
- 每次传输只需要 2 个 MCLK 时钟周期。
- 能够传输字节或字，以及混合的字节/字。
- 块尺寸最高可达 65 535 字节/字。
- 具有可配置的传输触发选择。
- 具有可选择的沿或电平触发传输。
- 具有 4 种寻址方式。
- 具有单字、块或者突发块传输模式。

### 12.2.9 数字 I/O

MSP430 器件具有多达 10 个数字 I/O 端口，P1 ~ P10。每一个端口都有 8 个引脚。每一个 I/O 引脚都可以单独配置输入或输出方向，每一个 I/O 行都可以单独地读出或写入。

端口 P1 和 P2 具有中断功能。P1 和 P2 I/O 行的每一个中断都可以单独进行使能设置，以确定在输入信号的上升沿或下降沿给出中断。所有 P1 I/O 行都可以作为一个中断向量的中断源，而所有 P2 I/O 行都可以作为另外一个中断向量的中断源。

数字 I/O 特性如下：

- 独立的可编程单个 I/O。
- 输入或输出的任意组合。
- 单独可配置的 P1 和 P2 中断。
- 独立的输入和输出数据寄存器。

### 12.2.10 Watchdog 定时器

Watchdog 定时器 (WDT) 模块的主要功能是，当发生软件问题时，作为一个受控的系统重启触发源。如果超过了选择的时间间隔，就会产生系统复位。如果在某个应用中不需要 Watchdog 功能，可以将模块配置为时间间隔定时器，可以按照选择的时间间隔产生中断。

Watchdog 定时器模块的特点如下：

- 4 个软件可选择的时间间隔。
- Watchdog 模式。
- 间隔模式。
- 访问 WDT 控制寄存器的口令保护。
- RST/NMI 引脚功能的控制。
- 可选择的时钟源。
- 可以被停止以节省功耗。
- WDT + 中的时钟故障保护特性。

在PUC( Power Up Clear, 上电清除)条件下, WDT模块被配置为Watchdog模式, 使用DCOCLK, 具有初始的32 768周期复位时间间隔。在初始复位时间间隔超过之前, 或者另外一个PWC产生之前, 用户必须建立、挂起或者清除WDT。当WDT被配置为工作在Watchdog模式下时, 使用错误的口令对WDTCTL进行写操作, 或者超过了选定的时间间隔, 都会触发一个PUC; PUC将WDT复位到它的默认条件, 并将RST/NMI引脚配置为复位模式。

### 12.2.11 定时器A和B

定时器A是一个16位的定时器/计数器, 有5个捕获/比较寄存器, 可以支持多个捕获/比较、PWM输出和时间间隔定时。定时器A还具有中断功能, 当溢出时, 计数器可以产生中断, 任意一个捕获/比较寄存器也可以产生中断; 功能框图如图12-51所示。

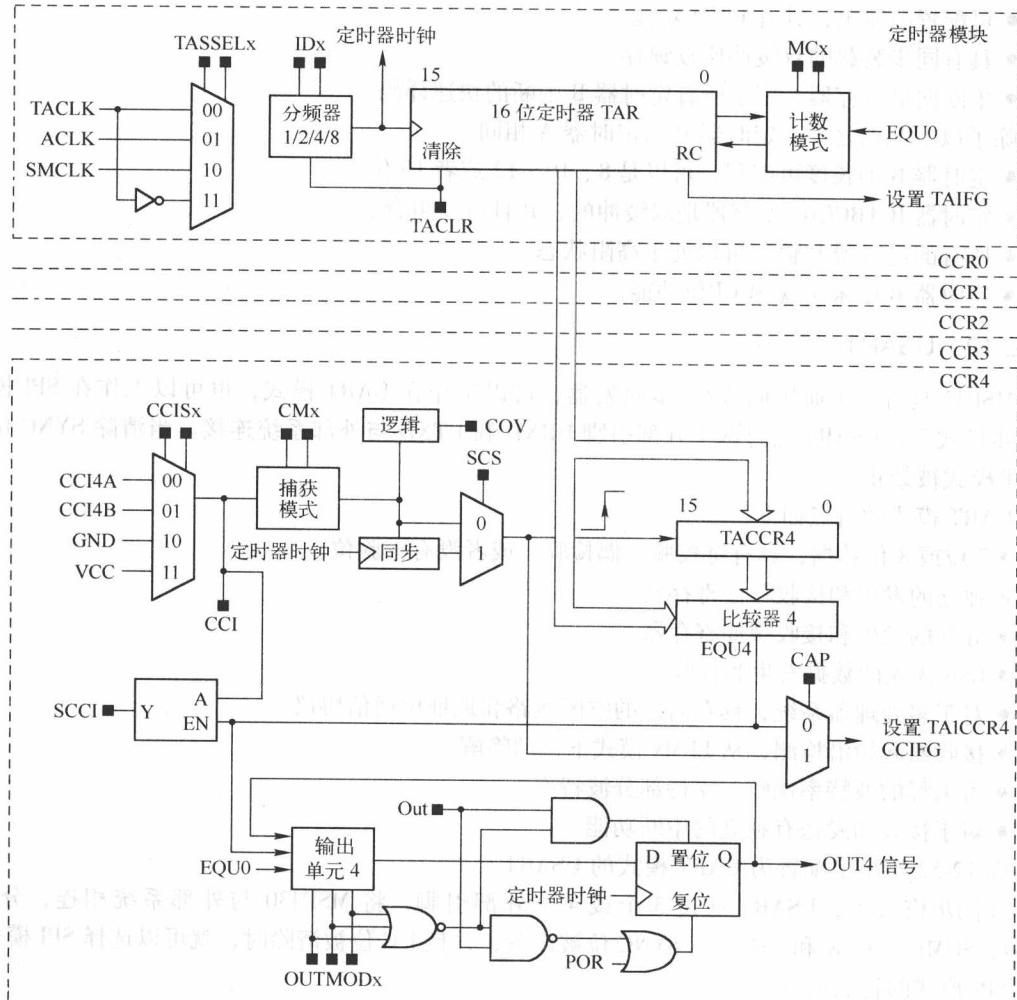


图12-51 定时器A框图(经德州仪器许可使用)

定时器A的特点如下:

- 异步16位定时器/计数器, 具有4种操作模式。
- 可选择、可配置的时钟源。
- 3个或5个可配置的捕获/比较寄存器。

- 可配置的输出，具有 PWM 功能。
- 异步输入和输出锁存。
- 中断向量寄存器，用于所有定时器 A 中断的快速译码。

定时器 B 是一个 16 位的定时器/计数器，有 3 个或 5 个捕获/比较寄存器，可以支持多个捕获/比较、PWM 输出和时间间隔定时。定时器 B 还具有中断功能，当溢出时，计数器可以产生中断，任意一个捕获/比较寄存器也可以产生中断。

定时器 B 的特点如下：

- 异步 16 位定时器/计数器，具有 4 种操作模式和 4 种可选择的长度。
  - 可选择、可配置的时钟源。
  - 3 个或 7 个可配置的捕获/比较寄存器。
  - 可配置的输出，具有 PWM 功能。
  - 具有同步装载的双缓冲比较锁存。
  - 中断向量寄存器，用于所有定时器 B 中断的快速译码。
- 除了以下几点之外，定时器 B 与定时器 A 相同。
- 定时器 B 的长度可编程，可以是 8、10、12 或者 16 位。
  - 定时器 B TBCCR<sub>x</sub> 寄存器是双缓冲的，并且可以组合。
  - 所有的定时器 B 输出可以处于高阻状态。
  - 定时器 B 中未实现 SCCI 位功能。

## 12.2.12 USART

MSP43 具有一个通用同步/异步收发器，可以工作在 UART 模式，也可以工作在 SPI 模式。在同步模式下，USART 通过两个外部引脚 URXD 和 UTXD 与外部系统连接。当清除 SYNC 位时，UART 模式被禁止。

USART 模式的特点如下：

- 7 位或 8 位数据，具有奇校验、偶检验，或者没有校验位。
- 独立的发生和接收移位寄存器。
- 分开的发生和接收缓冲寄存器。
- LSB 优先的数据发生和接收。
- 对于多处理器系统，具有内建的空闲线路和地址位通信协议。
- 接收器起始沿检测，从 LPMx 模式下自动唤醒。
- 可编程的波特率调整，支持部分波特率。
- 对于接收和发送有独立的中断功能。

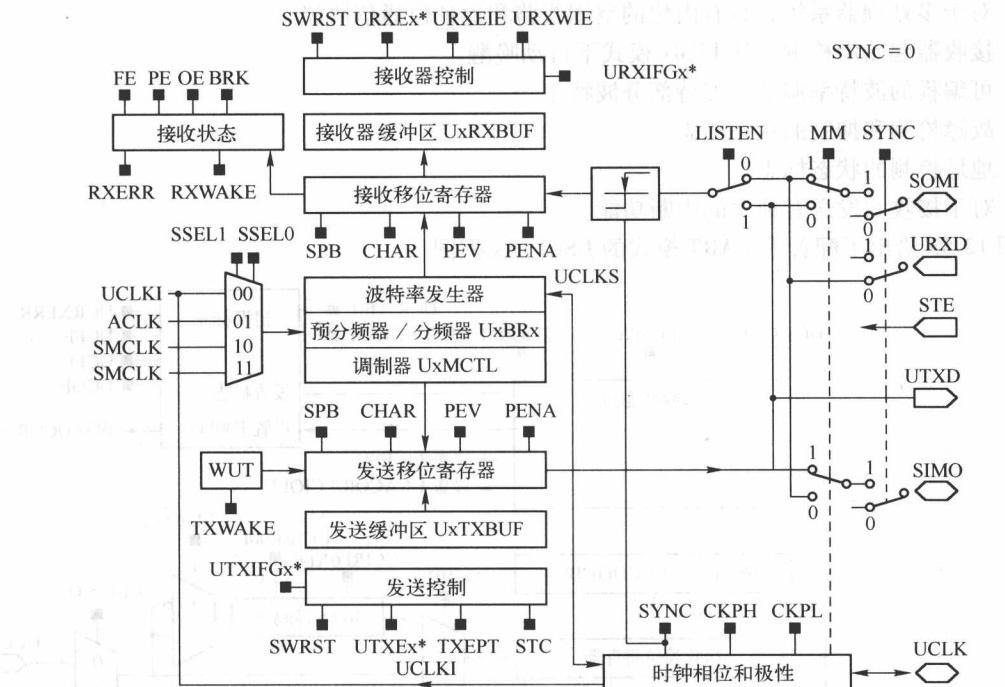
图 12-52 给出了配置为 UART 模式的 USART。

在同步模式下，USART 通过 3 个或 4 个外部引脚，将 MSP430 与外部系统相连，分别为 SIMO、SOMI、UCLK 和 STE。当 SYNC 位被设置，并且 I<sup>2</sup>C 位被清除时，就可以选择 SPI 模式。

SPI 模式的特点如下：

- 7 位或 8 位数据长度。
- 3 引脚和 4 引脚 SPI 操作。
- 主设备或从设备模式。
- 独立的发生和接收移位寄存器。
- 分开的发生和接收缓冲寄存器。
- 可选择的 UCLK 极性和相位控制。

- 在主模式下，UCLK 频率可编程。
- 对于接收和发送有独立的中断功能。



注：\* 关于 SFR 的位置，请参考特定器件的说明书。

图 12-52 USART 模式下的 USART (经德州仪器许可使用)

### 12.2.13 USCI

通用串行通信接口（USCI）模块支持多串行通信模式，不同的 USCI 模块支持不同的模式。每一个不同的 USCI 模块都以不同的字母命名，例如，USCI\_A 与 USCI\_B 不同，等等。如果在一个设备中实现多个同一种 USCI 模块，这些模块的名字用数字增量来表示。例如，如果一个设备中有 2 个 USCI\_A 模块，它们的名字为 USCI\_A0 和 USCI\_A1。

USCI\_Ax 模块支持如下情况：

- UART 模式。
- 针对 IrDA 通信调整脉冲。
- LIN 通信中自动波特率检测。
- SPI 模式。

USCI\_Bx 模块支持如下模式：

- I<sup>2</sup>C 模式。
- SPI 模式。

#### 1. UART 模式

在异步模式中，USCI\_Ax 模块通过 USCxRXD 和 UCAXTXD 2 个外部引脚，将 MSP430 与外部系统连接。当 UCSYNC 位被设置时，可以选择 UART 模式。

UART 模式的特点如下：

- 7 位或 8 位数据，具有奇校验、偶检验，或者没有校验位。
- 独立的发生和接收移位寄存器。

- 分开的发生和接收缓冲寄存器。
- LSB 优先或 MSB 优先的数据发生和接收。
- 对于多处理器系统，具有内建的空闲线路和地址位通信协议。
- 接收器起始沿检测，从 LPMx 模式下自动唤醒。
- 可编程的波特率调整，支持部分波特率。
- 故障检测和抑制的状态标志。
- 地址检测的状态标志。
- 对于接收和发送有独立的中断功能。

图 12-53 给出了配置为 UART 模式的 USCI\_Ax 框图。

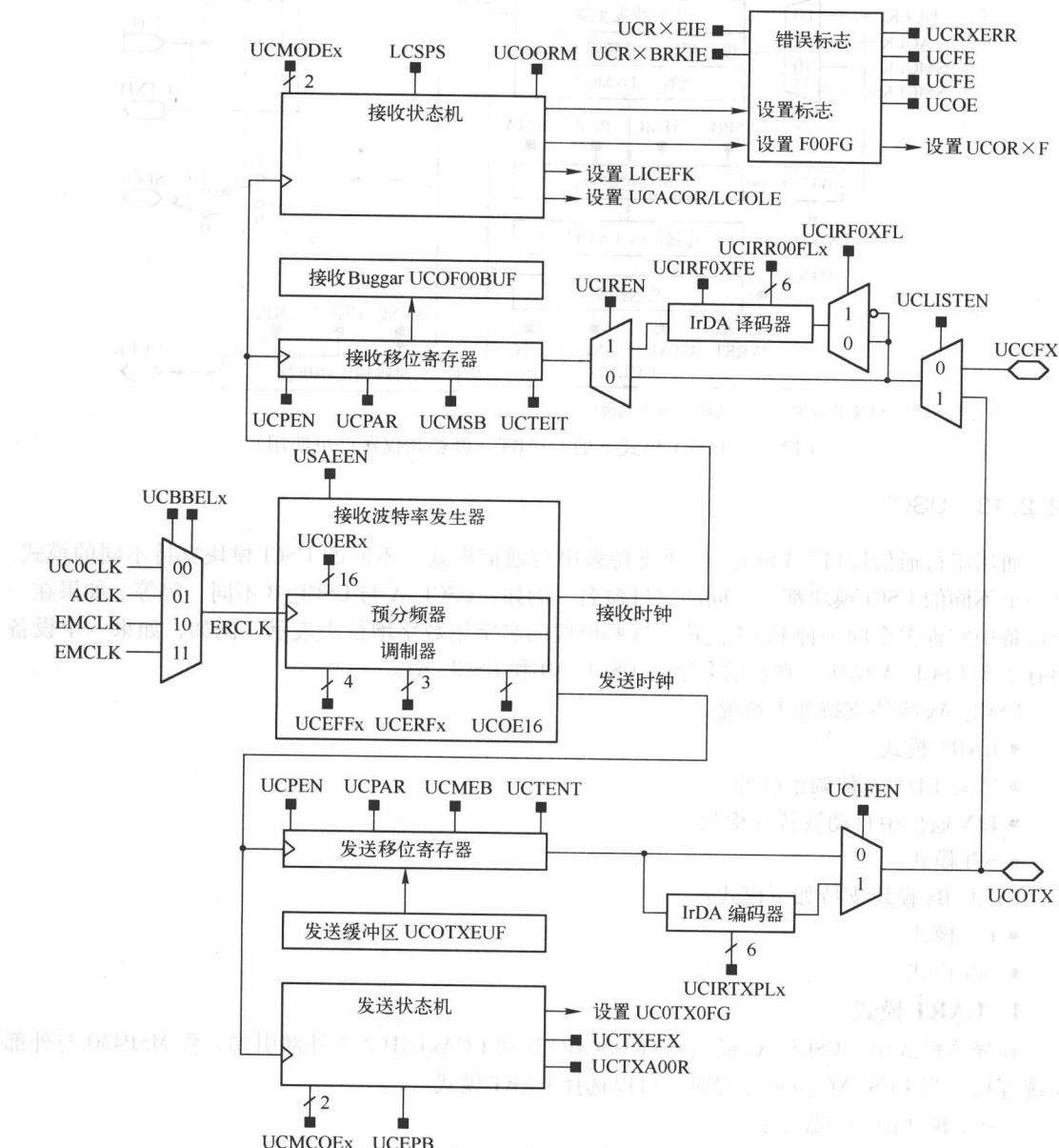


图 12-53 UART 模式下的 USCI (经德州仪器许可使用)

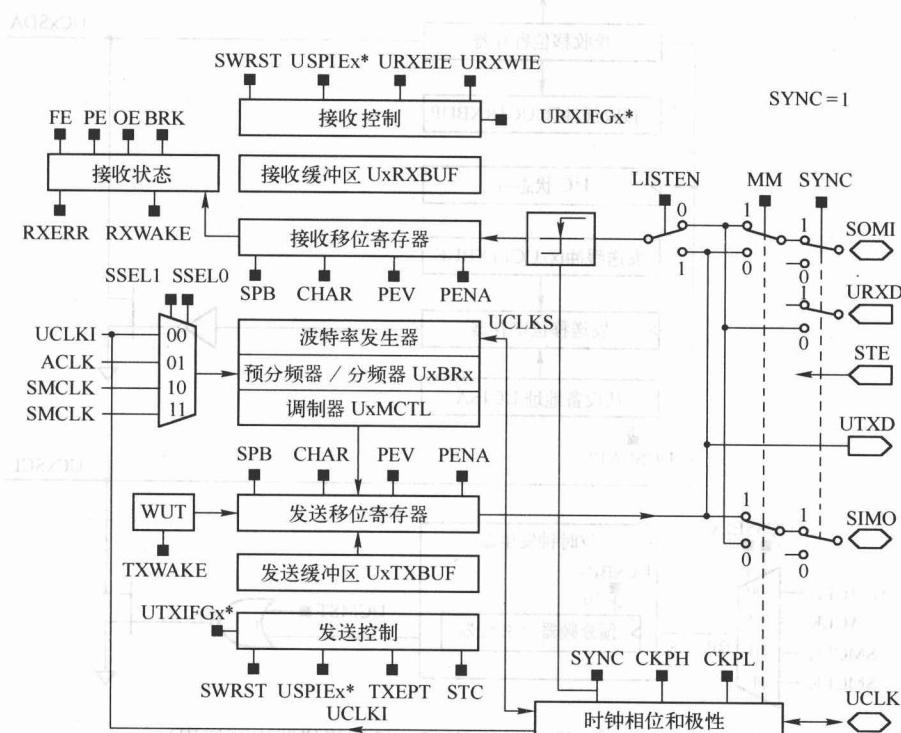
## 2. SPI 模式

在同步模式下，USART 通过 3 个或 4 个外部引脚，将 MSP430 与外部系统相连，这几个引脚分别为 SIMO、SOMI、UCLK 和 STE。当 SYNC 位被设置，并且 I<sup>2</sup>C 位被清除时，就可以选择 SPI 模式。

SPI 模式的特点如下。

- 7 位或 8 位数据长度。
- 3 引脚和 4 引脚 SPI 操作。
- 主或从模式。
- 独立的发生和接收移位寄存器。
- 分开的发生和接收缓冲寄存器。
- 可选择的 UCLK 极性和相位控制。
- 在主模式下，UCLK 频率可编程。
- 对于接收和发送有独立的中断功能。

图 12-54 给出了配置为 SPI 模式的 USCI。



注：\* 关于 SFR 的位置，请参考特点器件的说明书。

图 12-54 SPI 模式下的 USCI (经德州仪器许可使用)

## 3. I<sup>2</sup>C 模式

在 I<sup>2</sup>C 模式下，USCI 模块通过两线的 I<sup>2</sup>C 串行总线，在 MSP430 和 I<sup>2</sup>C 兼容的设备之间建立接口。连接到 I<sup>2</sup>C 总线的外部设备通过两线的 I<sup>2</sup>C 接口，串行地向 USCI 模块发送串行数据，或从 USCI 模块接收串行数据。

I<sup>2</sup>C 模式的特点如下：

- 与 Philips 半导体的 I<sup>2</sup>C 说明书 v2.1 版本相抑制。

- 7 位和 10 位的设备寻址模式。

- 广呼。

- 启动/重启/停止。

- 多主设备/接收设备模式发送。

- 最高 100 Kbit/s 的标准模式和最高 400 Kbit/s 的快速模式。

- 在主模式下，UCxCLK 频率可编程。

- 低功耗设计。

- 从接收器 START 检测，可以自动从 LPMx 模式唤醒。

- LPM4 中的从动工作。

图 12-55 给出了配置为 I<sup>2</sup>C 模式的 USCI。

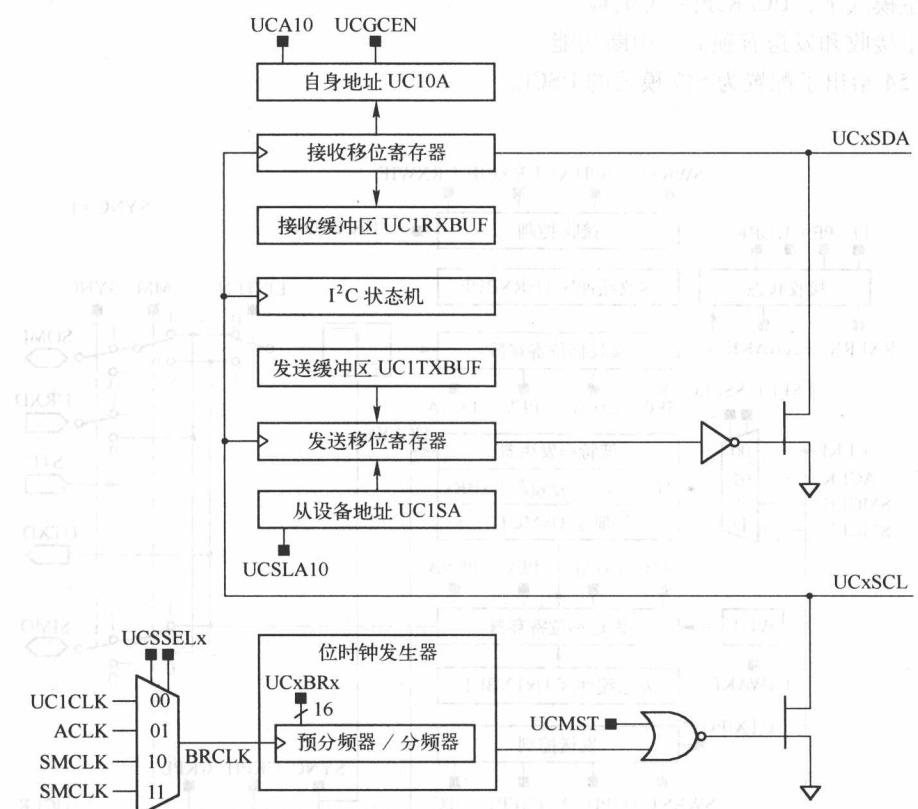


图 12-55 I<sup>2</sup>C 模式下的 USCI (经德州仪器许可使用)

## 12.2.14 ADC12 的功能

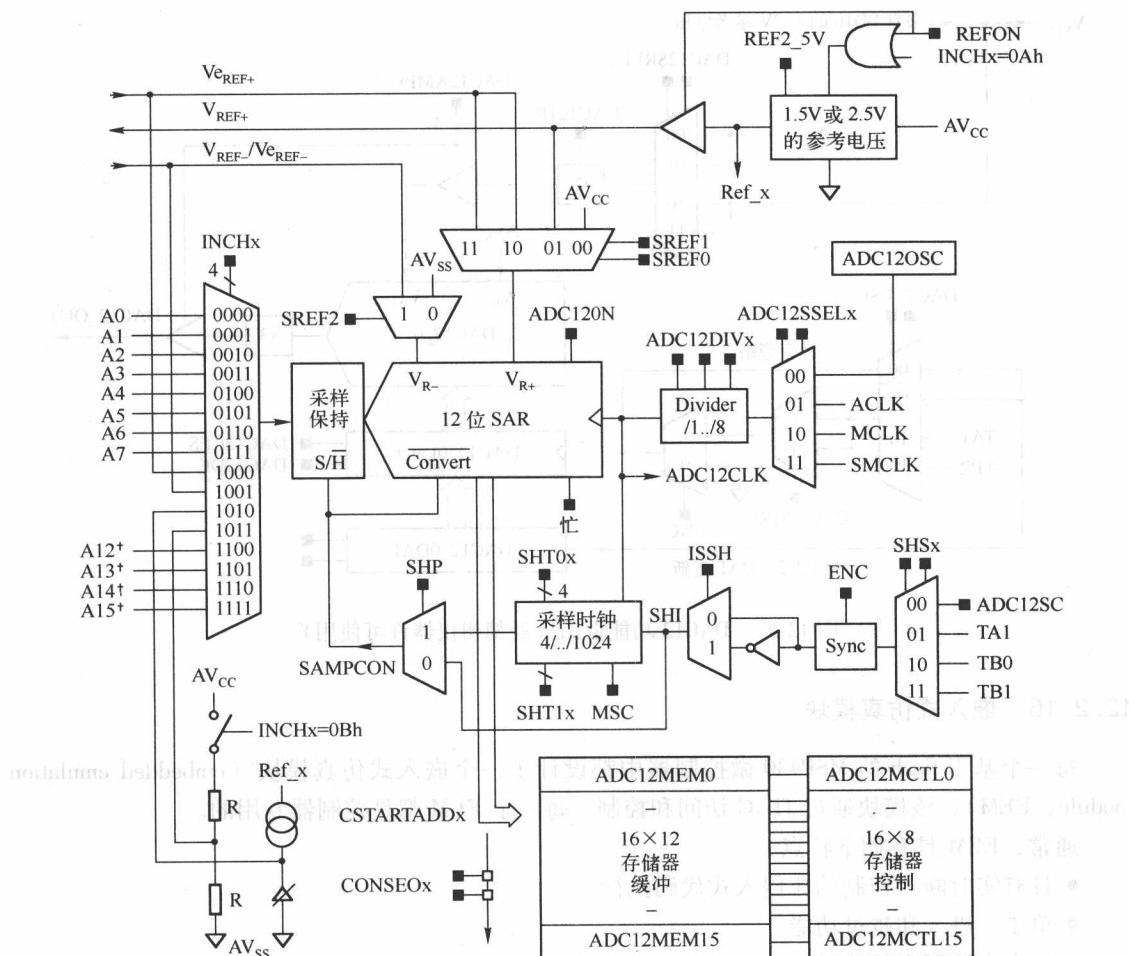
ADC12 模块支持快速 12 位模数转换。模块中有一个 12 位的 SAR 核、采样选择控制、参考发生器和一个 16 字的转换控制缓冲区。转换控制缓冲区最高支持 16 个独立的 ADC 采样，在不需要 CPU 干预的情况下，进行转换和存储。

ADC12 模块的特点如下：

- 最高转换漏码 12 位转换器。
- 软件或定时器控率大于 200 Kbit/s。

- 单调的、无制的采样周期可编程采样和保持。
- 软件、定时器 A 或定时器 B 对转换进行初始化。
- 软件可选择的片上参考电压产生 (1.5 V 或 2.5 V)。
- 软件可选择的内部或外部参考。
- 8 个独立的可配置外部输入通道 (对于 MSP430FG43x 和 MSP430FG461x 为 12 个)。
- 内部温度传感器、AVcc 和外部参考的转换通道。
- 对于正参考和负参考，都有独立的通道可选参考源。
- 转换时钟源可选。
- 支持单通道、重复单通道、序列和重复序列转换模式。
- 可以对 ADC 内核和参考电压独立断电。
- 中断向量寄存器可以对 18 个 ADC 中断快速译码。
- 16 个转换结果保存寄存器。

ADC12 的框图如图 12-56 所示。



\* 只针对 MSP430FG43x 和 MSP430FG461x。

图 12-56 ADC12 功能框图 (经德州仪器许可使用)

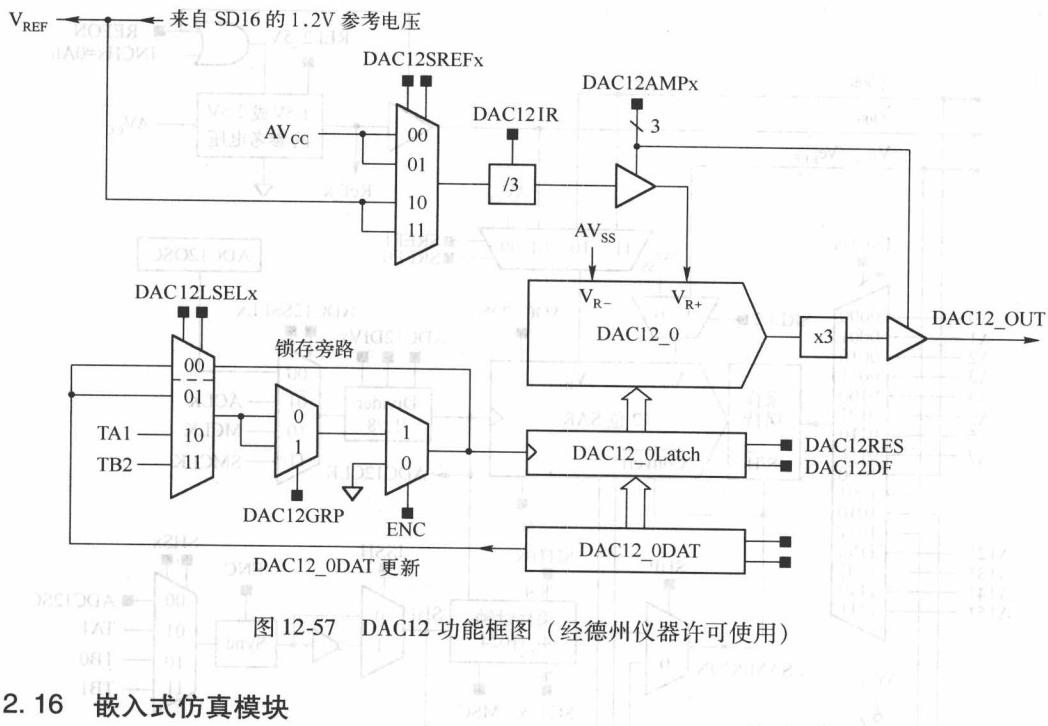
### 12.2.15 DAC12 模块

DAC12 模块是一个 12 位的电压输出 DAC。DAC12 可以被配置为 8 位或 12 位模式，可以与 DMA 控制器一起使用。当存在多个 DAC12 模块时，可以将它们组合在一起，执行同步更新。

DAC12 的特点如下：

- 12 位的单调输出。
- 8 位或 12 位的电压输出分辨。
- 可编程的稳定时间和功耗。
- 内部或外部参考选择。
- 二进制或 2 补数的数据模式。
- 自校准选择，用于偏移量修正。
- 具有对于多 DAC12 模块的自动更新功能。

图 12-57 给出了 DAC12 的框图。



### 12.2.16 嵌入式仿真模块

每一个基于 flash 的 MSP430 微控制器中都设计了一个嵌入式仿真模块（embedded emulation module, EEM），该模块通过 JTAG 访问和控制。每一个 EEM 都是控制器专用的。

通常，EEM 具有如下特点：

- 具有实时断点控制的非侵入式代码执行。
- 单步、进入和越过功能。
- 完全支持所有低功耗模式。
- 支持所有系统频率和所有时钟源。
- 在内存地址总线（memory address bus, MAB）或内存数据总线（memory data bus, MDB）上最多可以有 8 个（与微控制器有关）硬件触发/断点。

- 在 CPU 寄存器写操作上最多可以有 2 个（与微控制器有关）硬件触发/断点。
- 可以将 MAB、MDB 和 CPU 寄存器访问触发组合起来，形成 8 个（与微控制器有关）合成触发/断点。
- 触发序列（与微控制器有关）。
- 通过集成的跟踪缓冲区来保存内部总线和控制信号（与微控制器有关）。
- 在仿真停止过程中，针对定时器、通信外设和其他模块进行全局器件级或者基于单模块的时钟控制。

图 12-58 给出了最大的 4xx EEM 实现方式的简化框图。

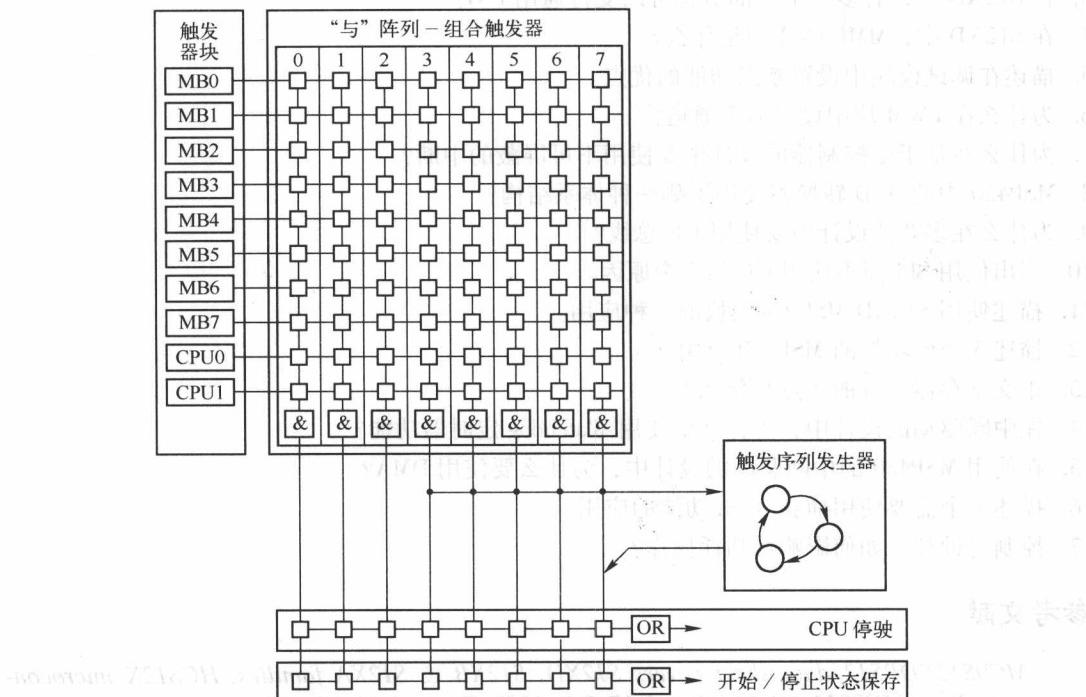


图 12-58 EEM 框图（经德州仪器许可使用）

## 触发

MSP430 系统中的 EEM 事件控制包括触发，是标志着某个事件发生的内部信号。这些触发可以用作简单的断点，也可以将两个或多个触发相结合，来检测复杂的时间，并触发除停止 CPU 之外的各种响应。

通常，触发可以用于控制 EEM 的下列功能块：

- 断点（CPU 停止）。
- 状态保存。
- 序列发生器。

触发共有 2 种——内存触发和 CPU 寄存器写触发。每一个内存触发模块可以进行独立的选择，将给定的数据与 MAB 或 MDB 进行比较。根据 EEM 的实现不同，比较可以是 =、≠、≥ 或 ≤；也可以使用屏蔽，将比较过程限定在某几位上。根据微控制器的不同，屏蔽可以按位，也可以按字节。除选择总线和比较之外，还可以选择在什么条件下触发起作用。这些条件包括读访问、写访问、DMA 访问和取指。

每一个 CPU 寄存器写触发块可以进行单独选择，将给定的值与写入选定寄存器中的值进行比较，比较可以是 =、 $\neq$ 、 $\geq$  或  $\leq$ ，也可以使用屏蔽将比较仅限在某几位上。将两种类型的触发组合在一起，可以形成复杂的触发。例如，当将某个特定的值写入用户指定的地址时，可以发出一个组合触发。

## 习题

1. 设置 XGATE 功能的主要原因是什么？
2. 为什么将 XGATE 的 R0 寄存器值固定为 0？
3. 在 S12XD 中，有多少个外部引脚可以支持通用 I/O？
4. 在 S12XD 中，MMC 的作用是什么？
5. 描述在调试模块中设置断点功能的优点。
6. 为什么在 PWM 块中设置多个通道？
7. 为什么在基于微控制器的设计中要使用不可屏蔽的中断？
8. MSP430 中的 A/D 转换器使用了哪一种体系结构？
9. 为什么在多芯片设计中要使用 I<sup>2</sup>C 总线？
10. 列出使用 SPI 而不使用 I<sup>2</sup>C 的 3 个原因。
11. 描述使用 S12XD MSCAN 模块的一种应用。
12. 描述 5 个低功耗的 MSP430 应用。
13. 正交寄存器文件的优点是什么？
14. 在中断驱动的设计中，为什么要使用 Watchdog 定时器功能？
15. 在使用 MSP430 的中断驱动的设计中，为什么要使用 DMA？
16. 描述一个需要使用捕获/比较功能的应用。
17. 控制时钟频率如何影响低功耗操作？

## 参考文献

*MC9S12XDP512 data sheet covers S12XD, S12XB & S12XA families, HCS12X microcontrollers, MC9S12XPD512, Rev. 2.17. July 2007.* Freescale Semiconductor.

*MSP430FG43X mixed signal microcontroller, SLAS380B.* June 2007. Texas Instruments.

*MSP430 ultra-low-power microcontrollers.* product brochure. SLAB043M. 2007. Texas Instruments.

*MSP430x4xx family user's guide, mixed signal products,* SLAU056G. 2007. Texas Instruments.

*S12XCPUV1 reference manual, HCS12X microcontrollers, S12XCPUV1, v01.01.* March 2005. Freescale Semiconductor.

*XGATE coprocessor fact sheet, XGATE COPROCFS REV 0.* Freescale Semiconductor.

## | 第13章 |

Embedded Microcontrollers and Processor Design

## 知识产权 SoC 核

- 本章目标：介绍基于 IP 的嵌入式核体系结构

- 学习内容：嵌入式处理器、存储器、I/O 和总线等模块的实现

关于可配置核设计和两个 IP 核的特点：

- 概述可配置核 SoC 设计。
- MIPS32 4K 系列嵌入式核。
- ARM10 嵌入式核。

### 13.0 SoC 概述

为了保持竞争力，片上系统（SoC）设计者必须跟上半导体技术的快速发展，新的通信、消费和计算机产品设计必须体现出功能、可靠性和带宽方面的快速提高；同时还必须考虑产品成本和功耗的下降。

这些发展都预示着对高集成度半导体的越来越多的需求，设计者一般使用寄存器传输级（register-transfer-level, RTL）硬件来实现数据密集处理功能，如图 13-1 所示。设计生产力差距、越来越高的纳米级半导体生产费用和上市时间的紧迫，都给了芯片设计者很大的压力，迫使他们设计出功能更复杂、速度更快、功耗更低的产品。

加速百万门级的 SoC 开发的一种途径是使用多个微处理器核承担当前转嫁给 RTL 技术的处理压力，如图 13-2 所示。尽管通用嵌入式处理器可以处理多个任务，它们通常缺少足够的代码，用于执行非常复杂的任务，例如音频和视频处理。因此，在 SoC 设计中，对 RTL 的使用出现了历史性的增长。

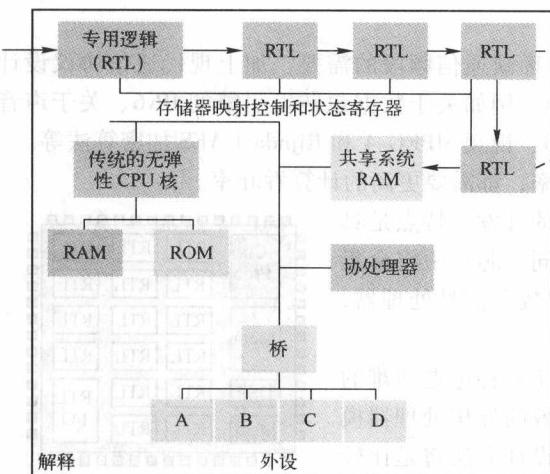


图 13-1 固定的 RTL SoC 实现

(经泰思立达公司许可使用)

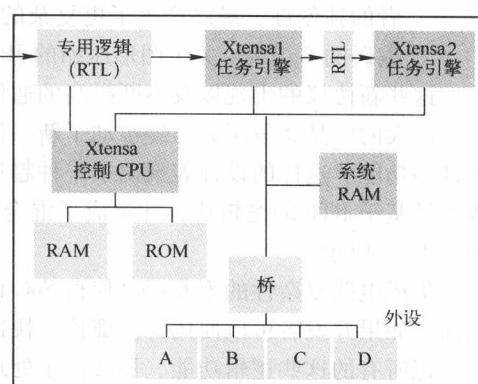


图 13-2 多处理器 Xtensa SoC 的实现框图

(经泰思立达公司许可使用)

开发人员可以配置一种新型的处理器，自动产生可扩展微处理器核，例如 Tensilica（泰思立达）的 Xtensa LX2；或者用户可更改的核，例如 MIPS 的 M4K，以产生需要的处理带宽，完成嵌

入式处理任务。由于这些可配置的处理器采样了固件，而没有采用 RTL 定义的硬件来实现它们的控制算法，对于同样的任务，设计者可以针对许多嵌入式 SoC 任务，比起使用基于 RTL 的硬件模块，可以更快、更容易地开发和校验基于处理器的任务引擎。

### 13.1 SoC 设计挑战

典型的深亚微米集成电路 (IC) 设计的一些特点给 SoC 设计团队提出了如下挑战。

- 在  $0.13\mu\text{m}$  标准单元的生产过程中，硅芯片密度超过了每平方毫米 100 000 门。因此，一个低成本面积为  $50\mu\text{m}^2$  芯片中可以设计 500 万个逻辑门。由于这样的高集成度成为可能，系统设计者总能找到一种途径，在给定的市场中发掘这种高集成度所带来的计算潜力。
- 在过去，硅生产量和设计自动化工具限制了实际的 RTL 块尺寸，小于 100 000 个门。改进的综合、布局布线和验证工具已经将这个数字大幅度提高，现在，这些工具的处理量可以高达 500 000 门。而现存的设计和验证工具并没有跟上硅生产量的发展速度，在实际生产中，限制了将数百万个门集成到一个 SoC 上。
- 典型逻辑块设计复杂性要求的增长比门数量的增加速度快得多，块数量的增加还远赶不上系统复杂性的要求，验证复杂性的发展也已经与门数量的发展不成比例。因此，许多最近开发了实际应用的设计团队都宣布，他们现在需要将 90% 的开发力量用于模块级或系统级的验证。
- 设计 Bug 的代价不断提高。工业分析家预测深亚微米集成电路掩膜的增加成本会很大，一个完整掩膜组的成本高达约 100 万美元，出现 Bug 的危险更会增加成本。在设计复杂的 SoC 时，需要更庞大的团队、更多的人员费用、更大的工程反复费用，一旦发生了设计 Bug，就会丧失巨大的收益率和市场份额。SoC 设计 Bug 能够毁掉一个公司，因此，能够减少这种 Bug 出现的设计方法，或者找到对这些 Bug 的低成本解决方法，都会很快获得收益。
- 所有嵌入式系统都包含大量软件。软件集成通常是系统开发过程的最后一步，而这一步通常会造成整个工程的延迟。分析家已经指出，对于新产品开发工程来说，更早、更快的硬件和软件验证是降低风险的关键方法。
- 标准的通信协议复杂性在迅速提高。保留稀缺通信频段的需要，加上现代通信协议设计者的独创性，最终产生了更复杂的新标准，例如关于数据包转发引擎的 IPv6、关于声音编码的 G. 729、关于图形压缩的 JPEG2000、视频 MPEG-4 和 Rijndael AES 加密算法等。

这些新协议的出现以及不断提高的通信位速率，都需要更高的计算吞吐率。

巨大的产品竞争压力已经推动了新一代 SoC 的开发，特点是具备许多功能。这样的设计表现出了一种趋势，在同一芯片中将会集成大量基于 RTL 的逻辑块，并集成了混合控制和数字信号处理器，如图 13-3 所示。

集成电路复杂性的不断提高使得 SoC 设计处于一种进退两难的局面。如果开发者可以使用许多廉价、快速和高效的异构处理器模块实现所有的这些逻辑功能，那么基于处理器的设计方法将是较好的选择。因为使用预先设计好并验证过的处理器核作为 SoC 的功能模块，可以将大部分的设计工作转化成对许多相对较小的软件块的编程工作。

这种设计 SoC 的方法可以在几分钟之内定位 Bug，而使用传统的设计方法，这个过程往往需要几个月的时间，因为修改和验证软件比修改 RTL 硬件要简单得多，尤其是当代码保存在片上

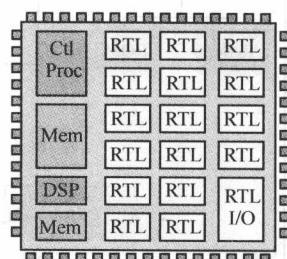


图 13-3 RTL 激增  
(经泰思立达公司许可使用)

RAM 时。然而不幸的是，对于大多数计算问题，通用处理器核在应用吞吐量、成本和功耗方面却远远不足。

同时，针对复杂功能和不断出现的标准设计定制的 RTL 逻辑，周期太长，并且一旦设计完成，就很难对逻辑进行修改。典型 RTL 块组成的内部描述如图 13-4a 所示，对问题的关键进行了近距离的观察；而图 13-4b 中给出了更加灵活的实现方式。

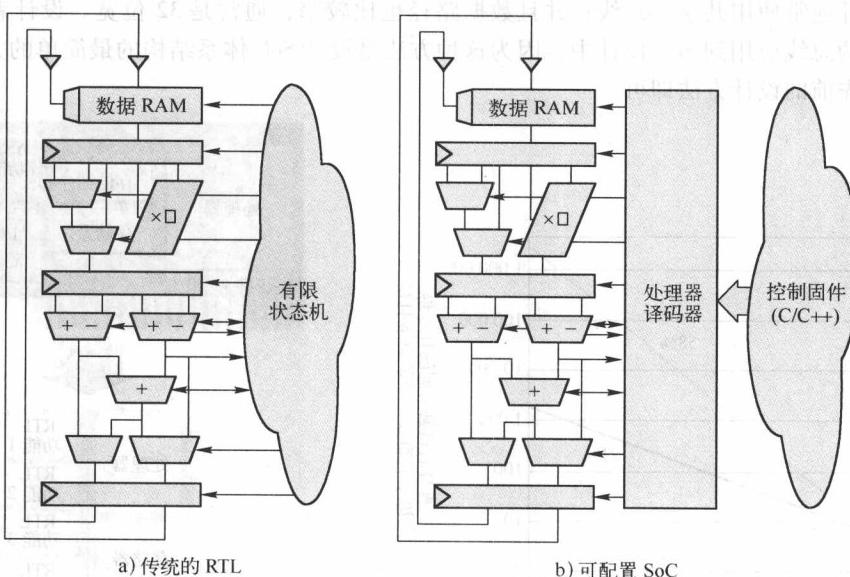


图 13-4 (经泰思立达公司许可使用)

在大多数 RTL 设计中，数据通路占用了最多的逻辑门。典型的数据通路可以是比较窄的 16 位或 32 位，或者可以是比较宽的数百位。数据通路的宽度通常由要处理的任务确定。数据通路中一般包括许多数据寄存器，用于表示中间的计算状态，并且还会包含大量的 RAM 块，或者与其他 RTL 块共享的 RAM 接口。这些基本数据通路结构体现了数据性质，并且在很大程度上独立于处理数据的特定算法细节。

与之相反，RTL 逻辑块的有限状态机中只包含控制细节。这种 RTL 块子系统捕获经过数据通路的数据序列的所有细微差别、所有异常的错误条件以及所有与其他模块的握手信号。状态机所需要的门数量比较少，但是由于它的复杂性，大多数设计和验证风险都来自于状态机。如果设计者在设计后期对一个 RTL 块进行修改，这种修改对状态机的影响将远大于对数据通路结构的影响，因此会增加设计风险。

可配置、可扩展的处理器是微处理器新的基本形式，可以减小状态机设计所带来的风险。通过将难以设计和验证的状态机逻辑块替换为预先设计好并验证好的处理器核和应用固件，可以实现这一点。

### 13.1.1 可配置处理器

根据摩尔定律，迅速增加的逻辑复杂性和工艺特征尺寸的降低，使得数百万门的设计成为可能。系统特性和功能方面的激烈产品竞争使得对这些先进半导体设计的需求越来越多。一种公认的 SoC 设计矛盾，存在于芯片复杂性的提高和逻辑设计工具生产力的提高，两者之间的差距在逐年增加，如图 13-5 所示。

此外，市场的趋势是需要高性能、低功耗的系统，这样促进了 SoC 设计数量的增加。一些应

用，例如长电池寿命手机、8 兆像素数码相机、快速价廉的彩色打印机、高清数字电视和 3D 视频游戏，都推动了设计资源的出现。如果不采取措施来缩短设计差距，将不可能把这些系统的增强版本设计推向市场。

如图 13-6 所示，传统的 SoC 设计模型非常类似于它的前身，即标准微处理器、内存和由专用集成电路（ASIC）实现的逻辑电路的板级组合。板级的芯片与芯片互联代价大，并且速度慢，因此板级设计通常使用共享的总线；并且数据路径也比较窄，通常是 32 位宽。设计者经常将这些相对限制的总线应用到 SoC 设计中，因为这种方法是设计 SoC 体系结构的最简单的方法，只需要重复使用先前的设计方法即可。

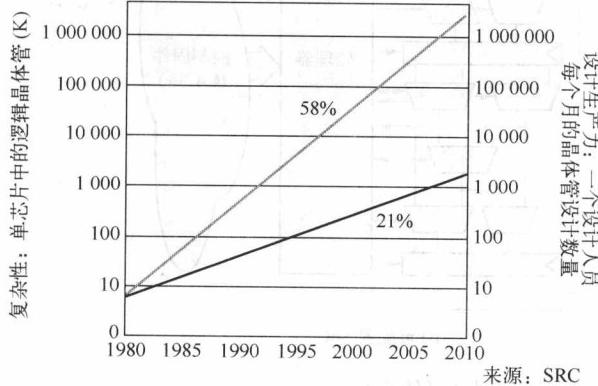


图 13-5 设计复杂性和设计生产力的提高

（经泰思立达公司许可使用）

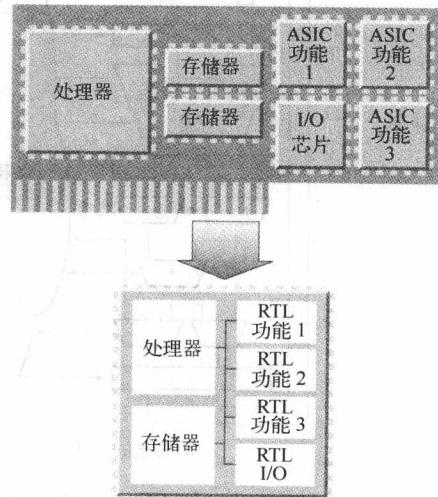


图 13-6 板级设计向 SoC 设计的转换

（经泰思立达公司许可使用）

将所有这些系统部件集成到一个硅片上，增加了时钟频率的最高值；并且与等价的板级设计相比，降低了功耗，系统可靠性和成本也得到了改善。这些好处可以说明 SoC 设计所带来的收益。然而，转向 SoC 综合并没有自动改变一个设计的组织或体系结构。因此，这些芯片的体系结构一般都继承了板级设计的设想、限制和折中。

### 13.1.2 SoC 综合

微处理器的出现和发展进一步约束了它们在传统 SoC 设计中的使用。最流行的嵌入式微处理器，尤其是 32 位体系结构的处理器，是直接从 20 世纪 80 年代的桌面计算机体系结构发展而来的，例如 ARM、MIPS、Freescale ColdFire 和 Intel Viiv 等。这些处理器是针对通用应用进行设计的，一般只支持最常用的数据类型，例如 8 位、16 位、32 位和 64 位整数。典型的处理器结构如图 13-7 所示，它们只支持最常用的操作，例如整数装载、保存、加、移位、比较和按位的逻辑操作等。

它们的通用特性使得这些处理器非常适用于在计算机系统上运行的各种应用。它们的体系结构在运行数据库、电子制表软件、PC 游戏和桌面出版等应用时，都表现出同样好的性能。然而，这些处理器都受到同一个瓶颈的困扰：它们

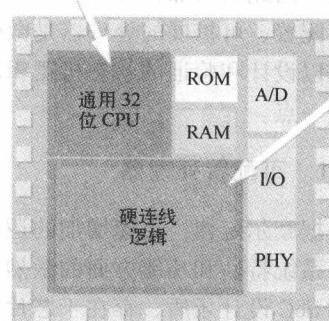


图 13-7 通用处理器

（经泰思立达公司许可使用）

对完全一般性的需求，也同样限制了它们执行处理未知数据类型的专用指令序列的能力。换句话说，通用处理器并不是针对处理任何给定嵌入式任务的专门数据类型而设计的，这样会影响效率。

与通用计算机系统相比，嵌入式系统包含各种组合，更加专用化。例如，数码相机必须运行许多复杂的图像处理任务，但是却永远不会执行 SQL 数据库查询；网络开关必须以光速处理复杂的通信协议，但是却不需要处理 3D 图像。

每个嵌入式应用的专门特性，对通用处理器运行数据量很大的嵌入式应用提出了两个问题。首先，许多嵌入式应用的功能和处理器的基本整数指令集和寄存器文件不相对应。由于这种不对应，当运行在通用处理器上时，重要的嵌入式应用需要更多的计算周期。其次，焦点更加集中的嵌入式器件不能很好地利用通用处理器的各种功能。这样由于分配给处理器的特定嵌入式任务并不需要一些功能，处理器资源就显得浪费。

许多嵌入式系统需要与周围环境密切交互，或者以很高的速度进行复杂的数据通信。高速运行的专用微处理器可以执行这些大数据量的任务，而当今 PC 中的处理器速度通常能达到几 GHz，因此这种处理是可以实现的。也就是说，用一个速度足够快的处理器处理，而不管它的成本或功能，可以解决任何问题。在图 13-8 中对这一点进行了解释。

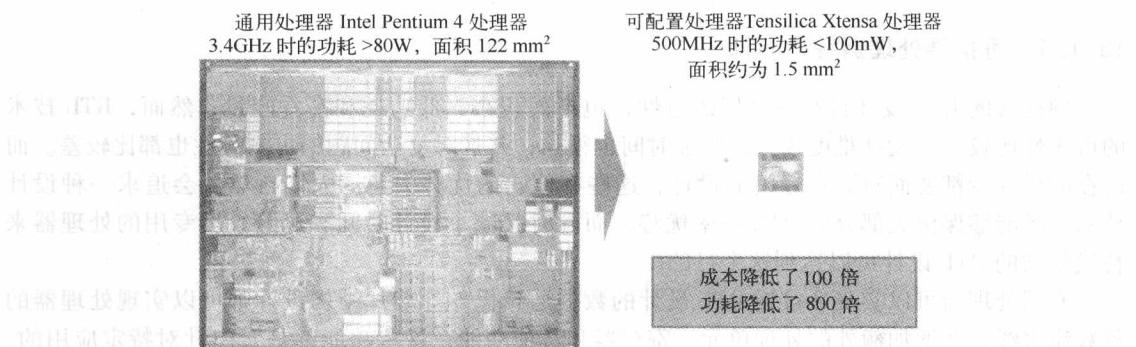


图 13-8 通用处理器与嵌入式处理器的比较（经泰思立达公司许可使用）

然而对于许多嵌入式任务，现在还不存在这样的处理器可以胜任这项工作，因为最快的处理器通常需要消耗过多的功耗和成本来满足嵌入式系统设计的要求。另一方面，嵌入式系统硬件设计者已经转向了硬连线电路，以处理这些数据密集型的任务。

在过去的 10 年中，逻辑综合和 ASIC 设计工具的发展使得 RTL 设计成为硬件开发人员的标准方法。与传统的晶体管级电路设计相比，基于 RTL 的设计效率要高得多，可以有效地发掘许多数据密集型问题内在的并行性。RTL 设计方法通常可以获得比通用处理器高几十或几百倍的性能提高。

### 13.1.3 可扩展处理器

与基于 RTL 的设计使用逻辑综合一样，可扩展处理器技术可以针对某个特定任务设计高速逻辑块。两种技术的区别在于，RTL 设计不仅实现了专用的数据通路，还用硬件实现了控制状态机；但是当用可扩展处理器来构建逻辑块时，设计者可以用硬件创建优化的数据通路，同时将控制功能完全用固件实现（见图 13-4a 和图 13-4b）。

一个有特色的、可配置的、可扩展的处理器包括一个处理器设计以及用于配置处理器的设计工具环境。如图 13-9 所示为 Xtensa 设计方法的基本要素，这个环境对基本的处理器设计有很

强的适应性，它允许系统设计者改变主要的处理器功能，将处理器调整为适应特定的应用需求。典型的可配置形式包括对内存、外部总线宽度和握手协议、一般处理器外设的增、减和调整。可配置处理器的超集，即可扩展处理器，允许应用开发者扩展处理器的指令集，并且可以扩充处理器的特性，这些特性在起初是设计者无法设想的。

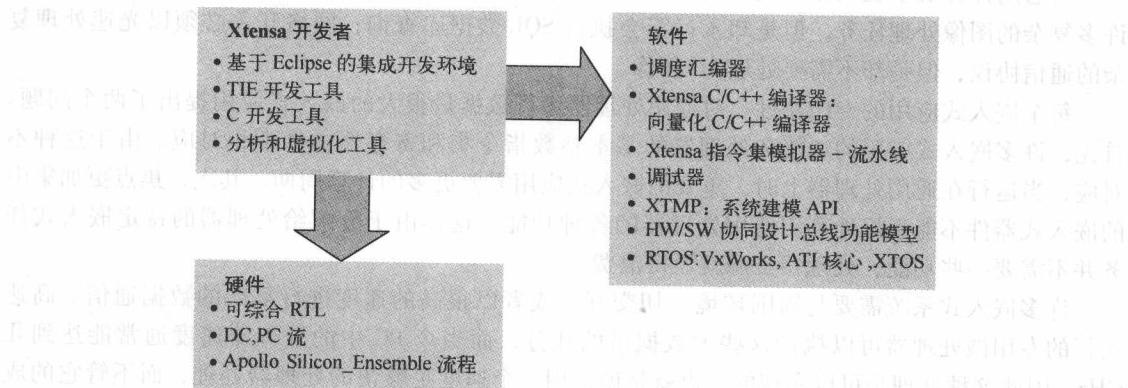


图 13-9 Tensilica Xtensa 设计环境（经泰思立达公司许可使用）

### 13.1.4 可扩展处理器替代 RTL

硬连线的 RTL 设计具有许多好的特性，包括面积小、低功耗和高吞吐量；然而，RTL 技术的可靠性比较差，设计难度较大，验证时间也很长，并且对复杂问题的可测量性也都比较差。而现在设计者都要面对数百万门的设计，这些都使它的优势失色。人们自然就会追求一种设计技术，既能够保留大部分的 RTL 效率优势，而同时又减小设计时间和风险。用专用的处理器来替代复杂的 RTL 设计可以达到这个目的。

专用处理器可以实现最接近 RTL 设计的数据通路操作。芯片结构设计师可以实现处理器的整数流水线，并增加额外的处理单元、寄存器和其他功能，这些功能都是专门针对特定应用的。例如使用 TIE 定义这种处理器时，设计者针对数据通路功能高级描述，以指令语义和编码的方式优化处理器，TIE 是 Tensilica 指令扩展语言，是 verilog 的变种。TIE 代码的实例如图 13-10 所示。

```

regfile vec 160 16v
operation MULA18.0 {inout vec acc,in vec m0,in vec m1}{{}
    wire[39:0]sum0 = m0[17: 0]*m1[17: 0]+acc[40: 0];
    wire[39:0]sum1 = m0[57:40]*m1[57:40]+acc[79:40];
    wire[39:0]sum2 = m0[97:80]*m1[97:80]+acc[119:80];
    wire[39:0]sum3 = m0[137:120]*m1[137:120]+acc[159:120];
    assign accum = {sum3,sum2,sum1,sum0};}

schedule mula {MULA18.0} {
    use m0 4;use m1 4;use acc 5;def acc 5;
}

```

图 13-10 TIE 代码实例（经泰思立达公司许可使用）

TIE 描述比 RTL 描述更加简洁，删除了所有时序逻辑，包括状态机描述、流水线寄存器和初始化序列。固件编程器可以通过同样的编译器和汇编器访问 TIE 中描述的新处理器指令和寄存器，编译器和汇编器中使用的是处理器的基本指令和寄存器集。固件使用处理器的常规取指、译码和执行机制用于控制处理器数据通路中的所有操作序列，开发者可以使用类似于 C 或 C++ 的高级语言来写这种固件。

用来替代 RTL 块的扩展处理器与传统的多数据通路 RTL 块具有相同的结构：深流水线、并行处理单元、专用的状态寄存器以及局部和全局存储器的宽数据通路。这些扩展的处理器可以实现同样高的计算吞吐量，并与典型的 RTL 设计一样支持低级数据接口。然而，扩展处理器数据通道的控制工作方式有很大的不同，基于处理器的任务引擎不是有硬连线的状态机，而是使用固件进行数据通路的控制。

### 13.1.5 清晰的控制方案

由于采用固件控制的状态转换，设计者并不固定地按周期控制处理器数据通路。他们将处理器执行的操作序列在固件中非常清楚地表示出来，如图 13-4b 所示。处理器使控制流在发生分支时有非常明确的决策，使得装载和保存操作中的存储器存取很明确，通用和专用计算操作序列中的计算序列也很明确。这种从硬连线状态机到固件程序控制的设计方法的转换具有如下优势。

- 灵活性。芯片设计者、系统构建者和（适当的话）终端用户可以通过只修改固件来改变模块的功能。
- 基于软件的开发。开发者可以使用复杂的、低成本的软件开发方法实现大多数的芯片特性。
- 更快、更完全的系统建模。RTL 的仿真速度是很慢的，对于一个千万门的设计，即使最快的基于软件的逻辑仿真器，其仿真速度也不会超过每秒几个周期。而另一方面，扩展处理器的固件模拟每秒可以运行几十万个周期。
- 控制和数据的统一。没有一个现代系统会是单纯用硬连线逻辑实现的，这样的系统通常包括一个处理器和一些软件。将先前用 RTL 来完成的功能转换成用处理器实现，可以消除控制和数据处理之间的区别。
- 缩短上市时间。将关键的功能从 RTL 实现方式转换为专用处理器引擎，可以简化 SoC 设计，加速系统建模以及硬件设计。基于固件的引擎易于标准化某些变化，因为硬件设计过程与详细的产品需求的最终实现是分开的。
- 提高设计者的生产力。最重要的是，将基于 RTL 的设计转换为使用预先设计好的、一次生成即保证正确的专用处理器，由于减小了 RTL 开发和验证的资源需求，而提高了设计团队的生产力。基于处理器的 SoC 设计方法可以极大地降低庞大逻辑 bug 的风险，并且当测试发现 bug 后，也很容易解决。

尽管有上述很多优势，但专用处理器并不是所有设计的最佳选择。因为存在如下 3 种例外情况。

- 小的、固定状态机。有些逻辑任务比较简单，不用处理器来完成。一些位串引擎，例如简单的通用异步收发器就属于这样的例子。
- 简单的数据缓冲。有些逻辑任务只是进行存储控制。处理器中的存储器操作可以模拟先进先出控制器，这样做需要随机存取存储器和一些包裹逻辑来协助完成，但是如果采用 FIFO，将会又快又简单。
- 超深流水线。一些计算问题非常规整，状态机控制比较简单，单个超深流水线就可以很好地解决这类问题。一些常见的例子（3D 图形和磁盘读通道芯片）有时具有几百个时钟站深度的流水线，可以用专用处理器来控制这样的流水线，但是逐条指令的控制对这些应用的控制效果并不会太好。

除了上述一些情况之外，固件程序控制的优势也使之成为一种明智的设计选择。随着时间的推移，从软件到硬件逻辑的功能转移正是对这种公认现象的体现。过去针对协议标准的设计开发过程中，通常采用基于处理器的方法，即使明显可以只采用简单的逻辑就可以实现的简单

标准也是如此。采用这种方式的一些标准包括流行的多媒体数字信号编解码器（如 MPEG-2）、3G 无线协议（如 W-CDMA）以及加密保密算法例如 SSL 和三元 DES 加密。

然而，基于软件的开发和基于 RTL 的开发之间在性能和设计难易程度上的巨大差异，已经限制了这种功能转移的发展。可配置和可扩展的转移处理器的出现开辟了一种新的设计途径，能够快速、简单地对新的协议和标准进行开发和精炼；同时在硅片面积和功耗方面具有足够的优势，可以进行大量的展开。

## 13.2 MIPS32 4K 处理器核系列

来自 MIPS 科技的 MIPS32 4K 处理器核是一种高性能、低功耗的 32 位 MIPS RISC 核，是针对片上系统应用而设计的。该处理器核（见图 13-11）是面向半导体生产公司、ASIC 开发人员和系统 OEM 设计的，设计者都希望将自己的逻辑和外设快速地与高性能 RISC 处理器进行集成。这些核是完全可综合的，具有很强的灵活性，它们可以适用于多种处理过程，易于在片上系统设计中进行集成。这样就使得开发人员将他们的注意力放在如何使产品的特性满足特定用户的需求上。

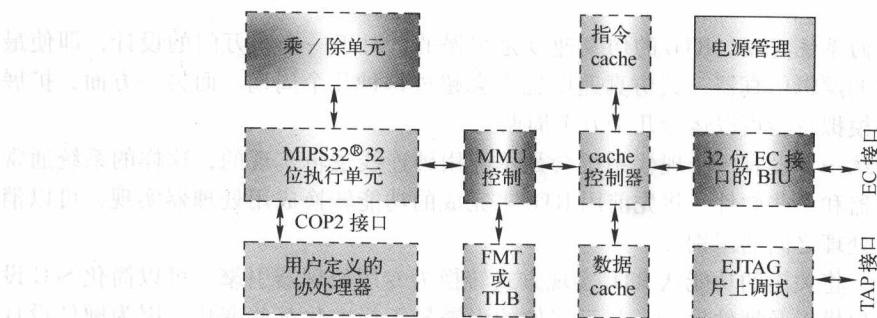


图 13-11 MIPS 4KE 系列框图（经 MIPS 许可使用）

4KE 系列有三个成员，分别是 4KEc、4KEm 和 4KEp 核。这些核兼有 MIPS 科技 R3000 和 R400 处理器的特点。这三种核的主要区别在于乘除单元（multiply-divide unit, MDU）和内存管理（memory management unit, MMU）的类型。

4KEc 核中设计了一种全相连的、基于旁路转换缓冲（translation lookaside buffer, TLB）的 MMU 和流水化的 MDU。

4KEm 核在 MMU 中设计了一种直接映射（fixed mapping, FM）机制，与 4KEc 核中所使用的基于 TLB 的方法相比，更小、更简单；同样也采用了流水化的 MDU（与 4KEc 核一样）。

4KEp 核中的 MMU 是基于 FM（固定映像）的（与 4KEm 核一样），但是它的 MDU 是非流水化的。指令和数据 cache 是完全可编程的，大小为 0~64K 字节。此外，每一个 cache 都可以按照直接相连、两路、三路或四路组相连。当发生 cache 失效时，装载过程被阻塞，直到第一个关键字取出为止。在其余字写入 cache 的过程中，流水线恢复执行。虚拟索引功能可以在产生地址的同一个时钟周期内对 cache 进行索引，而不必等待 TLB 中虚拟到物理地址的转换完成。

所有的核都执行 MIPS32 指令集体系结构（ISA）。MIPS32 ISA 既包括 MIPS II 的所有指令，也包括乘累加、条件转移、预取、等到和 0/1 检测指令。4KEc 核的 R4000 风格 MMU 中有一个四条目指令 TLB（ITLB）。一个四条目数据 TLB（DTLB）和一个 16 双条目的联合 TLB（JTLB），页大小是可变的。

4KEm 和 4KEp 处理器核采用的是简单的固定映像机制，地址空间的映像由 CPO 配置（选择

0) 寄存器中的位来确定。4KEc 和 4KEm 乘加单元 (MDU) 支持最大的输出率，在每个时钟周期中，执行一个  $32 \times 16$  乘 (MUL/MULT/MULTU)、乘加 (MADD/MADDDU) 或者乘减 (MSUB/MSUBU) 操作，或者每隔一个周期执行一个  $32 \times 32$  的 MUL、MADD 或 MSUB 操作。

处理器核中还设计了基本的增强 JTAG (EJTAG)，可以对 CPU 进行运行控制，有停止、单步和重启操作，并且可以通过 SDBBP 设置软件断点。其他的 EJTAG 特性，例如指令和数据虚拟地址硬断点、通过测试访问端口 (TAP) 与外部 EJTAG 探测器连接以及 PC/数据跟踪等，都是可以选择的。

### 13.2.1 4KE 系列的主要特点

1) 32 位地址和数据通路。

2) MIPS32 可兼容的指令集：

- 所有的 MIPSII 指令；
- 乘加、乘减指令 (MADD、MADDDU、MSUB、MSUBU)；
- 乘法指令 (MUL)；
- 0/1 检测指令 (CLZ、CLO)；
- 等待指令 (WAIT)；
- 条件移动指令 (MOVZ、MOVN)；
- 预取指令 (PREF)。

3) MIPS 16e 专用扩展：

- 32 位指令的 16 位编码，以提高代码密度；
- 特殊的 PC 相关指令，提高装载地址和常数的效率；
- 数据类型转换指令 (ZEB、SEB、ZEH、SEH)；
- 简洁的跳转 (JRC、JALRC)；
- 堆栈结构，建立和消除“宏”指令 (SAVE 和 RESTORE)；
- 用户自定义指令 (对这项功能的使用需要额外的许可)；
- 将可选的用户自定义指令添加到 MIPS32 指令集中 (作为创建时间的一个选项)；
- 单周期或多周期指令；
- 原操作数来自寄存器或立即数；
- 目的操作数在寄存器。

4) 可编程的 cache 大小：

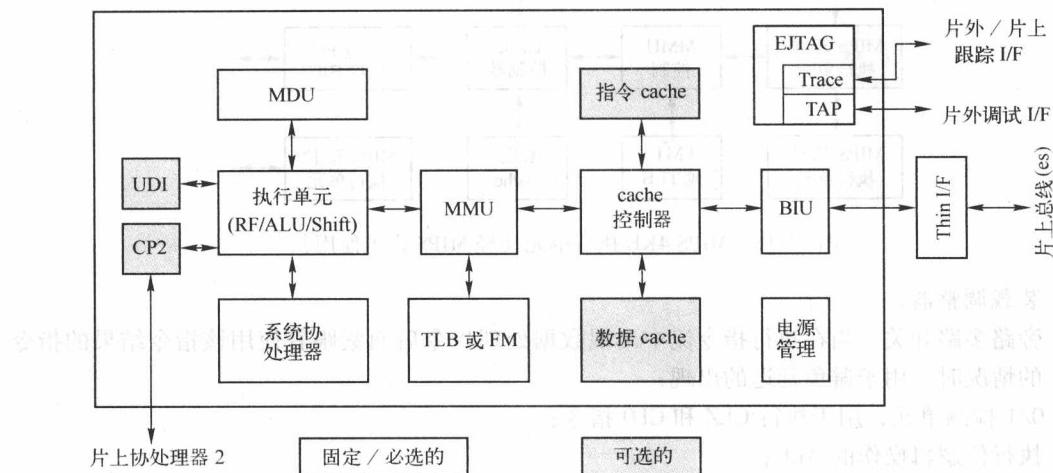
- 单独可配置的指令和数据 cache；
- 大小为 0~64K 字节；
- 直接映像或两路、三路、四路组相连映像；
- cache 失效被阻塞，直到关键字出现为止；
- 支持写分配和写回，以及具有或不具有写分配的写直达；
- 128 位 (16 字节) cache 行大小，一个字为一个区，适用于标准的 32 位宽的单端口 SRAM；
- 虚拟索引，物理标示；
- 支持 cache 行锁定；
- 非阻塞预取；

5) 中间结果暂存器 RAM 支持：替代指令 cache 和/或数据 cache 的一种方式；

- 替代指令 cache 和/或数据 cache 的一种方式；

- 最大 20 位的索引 (1M 地址)。
  - 与中间结果暂存器端口相连接的内存映像寄存器可以用作协处理器接口。
- 6) R4000 风格的特权资源体系结构:
- 用于实时定时器中断的计数/比较寄存器;
  - 用做软件断点的指令和数据观察寄存器;
  - 分开的中断异常向量。
- 7) 可编程的内存管理单元 (只针对 4KEc 核):
- 16 个双条目 MIPS32 风格的 JTLB, 页大小可变;
  - 4 条目的指令 TLB;
  - 4 条目的数据 TLB。
- 8) 可编程的内存管理单元 (只针对 4KEm 和 4KEp 核):
- 固定影响 (没有 JTLB、ITLB 或 DTLB);
  - 采用寄存器位的地址空间映像。
- 9) 简单的总线接口单元 (bus interface unit, BIU):
- 所有的 I/O 都完全寄存;
  - 分开的单向 32 位地址和数据总线;
  - 2 个 16 字节折叠式写缓冲。
- 10) 全功能的协处理器 2 接口:
- 几乎所有 I/O 都完全寄存;
  - 分开的单向 32 位地址和数据总线;
  - 支持协处理器条件转移;
  - 处理器和协处理器双向和寄存器数据传输;
  - 直接内存与协处理器双向寄存器数据传输。
- 11) 乘除单元 (4KEc 和 4KEm 核):
- 最大发送率为每周期一个  $32 \times 16$  乘法;
  - 最大发送率为每两个周期一个  $32 \times 32$  乘法;
  - 除法控制, 最小 11 个、最大 34 个时钟周期;
  - 时隙除法。
- 12) 乘除单元 (4KEp 核):
- 迭代的乘法和除法; 每条指令 32 个或更多个周期;
  - 电源控制;
  - 没有最小频率;
  - 掉电模式 (由 WAIT 指令触发);
  - 支持软件控制的时钟分频。
- 13) EJTAG 调制支持:
- 开始、停止和单步 CPU 控制;
  - 通过 SDBBP 指令设置软件中断;
  - 对虚拟地址可选的硬件断点;
  - 4 条指令和 2 个数据断点, 2 条指令和 1 个断点, 或者没有断点;
  - 可选的测试访问端口 (TAP), 用于高速下载应用程序代码;
  - 可选的 EJTAG 跟踪硬件, 对执行代码进行实时跟踪。
- 所有 MIPS 核既包含必选的功能块, 也包括可选的功能块。在图 13-12 中, 阴影表示的功能

块是4KE核中必选的，也就是说，如果要与MIPS兼容，这些模块是必须要设计的，可选的功能块可以根据实际应用添加到核中。



MIPS 核必选的功能块包括：

- 执行单元；
- 乘除单元 (MDU)；
- 系统控制协处理器 (control coprocessor, CP0)；
- 内存管理单元 (MMU)；
- cache 控制器；
- 总线接口单元 (BIU)；
- 电源管理。

MIPS 核可选的功能块包括：

- 指令 cache (I-cache)；
- 数据 cache (D-cache)；
- 增强的 JTAG (EJTAG) 控制器；
- 协处理器 2 接口 (CP2)；
- 用户自定义指令 (user-defined instructions, UDI)。

在4KEc核中，MMU通过TLB来实现；在4KEm和4KEp核中，MMU通过固定映像来实现。

### 13.2.2 执行单元

处理器核的执行单元实现了一种 load-store 体系结构和自主乘除单元（见图 13-13），其中，算术逻辑单元 (ALU) 的操作（逻辑、移位、加、减）是单周期的。处理器中共包含 32 个 32 位的通用寄存器 (general-purpose register, GPR)，用于标量的整数操作和地址计算。寄存器文件有两个读端口和一个写端口，并且为了减小流水线中的操作延迟，都是完全旁路的。

MIPS4KE 执行单元包括：

- 32 位的加法器，用于计算数据地址；
- 地址单元，用于计算下一条指令地址；
- 分支判断和分支目标地址计算逻辑；

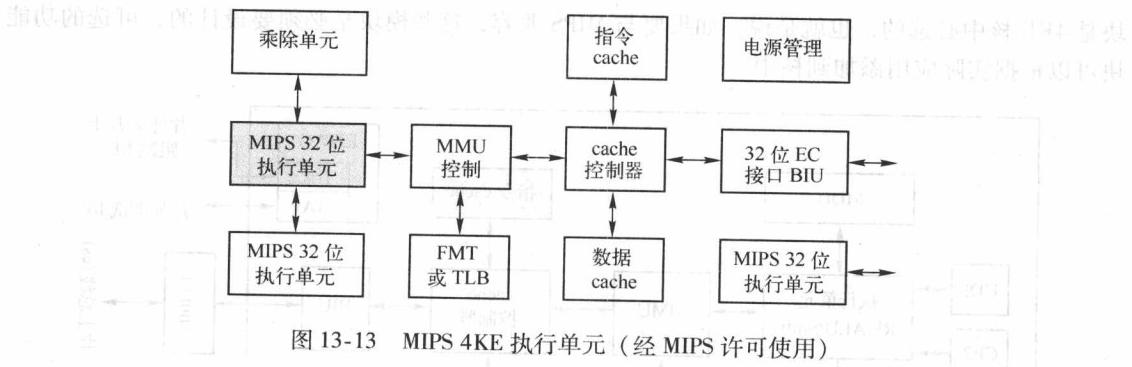


图 13-13 MIPS 4KE 执行单元 (经 MIPS 许可使用)

- 装载调整器；
- 旁路多路开关，当在执行指令流中出现数据处理指令后面紧跟着使用该指令结果的指令的情况时，用于避免延迟的出现；
- 0/1 检测单元，用于执行 CLZ 和 CLO 指令；
- 执行位逻辑操作的 ALU；
- 移位器和保存定位器。

### 13.2.3 乘除单元 (MDU)

MDU 的作用是执行乘法和除法操作（见图 13-14）。在 4KEc 和 4KEm 处理器中，MDU 包括  $32 \times 16$  的布斯编码乘法器、结果累积寄存器 (HI 和 LO)、乘法和除法状态机，以及执行这些功能需要的所有多路器和控制逻辑。流水化的 MDU 可以在每个周期中执行一个  $16 \times 16$  或  $32 \times 16$  的乘法操作，可以在每 2 个周期中执行一个  $32 \times 32$  的乘法操作。

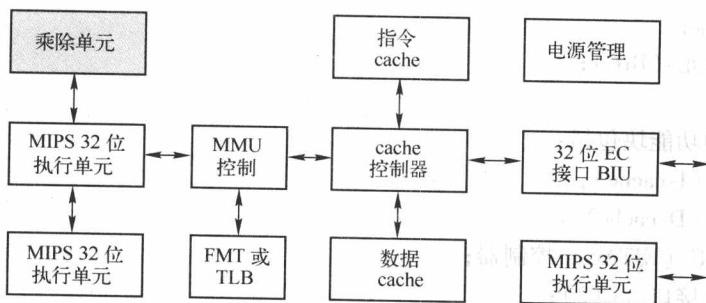


图 13-14 MIPS 4KE 乘除单元 (经 MIPS 许可使用)

处理器中还设计了适当的互锁功能，以避免出现背靠背的 32 位操作。除法操作中设计了一种简单的每周期 1 位的迭代算法，在最差的情况下，需要 35 个时钟周期才能完成。在算法的早期，须先检测被除数的符号扩展。如果被除数的实际大小是 24 位、16 位或 8 位的，除数将跳过 32 次迭代中的 7 位、15 位或 23 次迭代。当一条除法操作正在执行时，如果示图发送另外一个 MDU 指令，将会导致流水线的停顿，直到当前的除法操作完成为止。

在 4KEp 处理器中，非流水化的 MDU 包括一个 32 位的全加器、结构保留寄存器 (HI 和 LO)、一个组合的乘/除状态机以及完成这些操作需要的所有多路器和控制逻辑。执行乘法操作需要 32 个周期，采样每周期 1 位的迭代算法、触发操作也使用每周期 1 位的迭代算法 (非 early-in)，需要 35 个周期才能结束。当一条除法操作正在执行时，如果试图发送另外一个 MDU 指令，将会导致流水线的停顿，直到当前的除法操作完成为止。

所有的核中还设计了另外一条乘法指令，即 MUL。在该指令中，乘法操作的低 32 位存储在寄存器文件中，而不是存储在 HI/LO 寄存器对中。处理器中没有使用 MFLO 指令，该指令在对 LO 寄存器操作时使用，并且支持多目的寄存器；对于乘法操作比较多的应用，可以提高吞吐量。执行乘加和乘减操作时，会使用两个指令，分别是乘加（MADD/MADDU）指令和乘减（MSUB/MSUBU）指令。MADD 指令将两个数相乘，然后将结果与 HI 和 LO 寄存器中的当前值相加。类似地，MSUB 指令将两个操作数相除，然后用 HI 和 LO 寄存器中的值减去除法结果。MADD/MADDU 和 MSUB/MSUBU 操作经常在数字信号处理算法中使用。

在 MIPS 体系结构中，CP0 负责的操作包括虚实地址转换、cache 协议、异常控制系统、处理器的诊断功能、操作模式选择（内核模式还是用户模式）以及中断的使能/禁止。读取 CP0 寄存器的值，可以配置一些信息，例如 cache 大小、组相联选择和 EJTAG 调试特点等。

### 13.2.4 内存管理单元 (MMU)

每一个处理器核中都包括一个 MMU，用做执行单元与 cache 控制器的接口（如图 13-15 所示）。尽管 4KEc 核采用的是一种 32 位体系结构，根据 MIPS32 体系结构的定义，MMU 是根据 64 位 R4000 系列中的 MMU 结构来设计的。

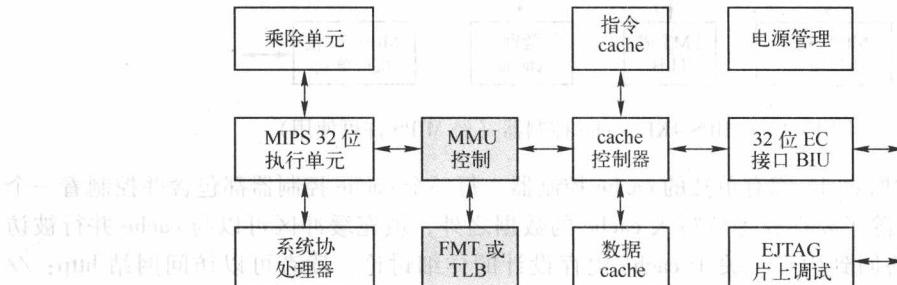
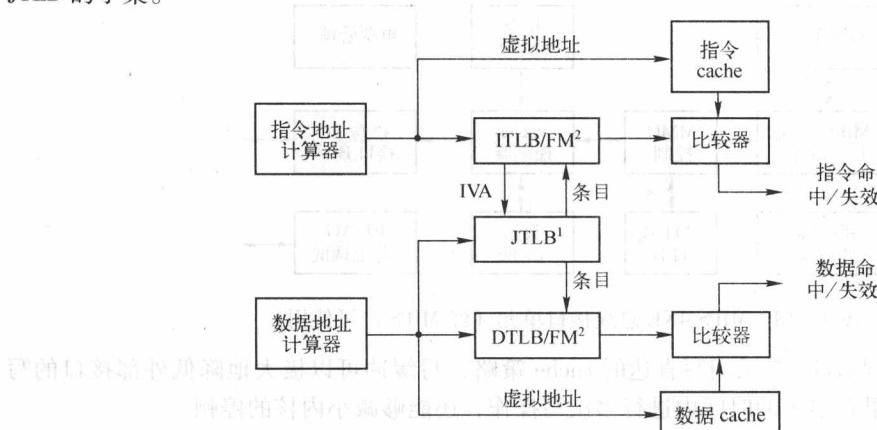


图 13-15 MIPS 4KE 内存管理单元 (经 MIPS 许可使用)

4KEc 核的 MMU 是基于 TLB 的，如图 13-16 所示。TLB 包括 3 个转换缓冲区：一个 16 位双条目全相联联合 TLB (JTLB)，一个 4 条目全相联指令 TLB (ITLB) 和一个 4 条目全相联数据 TLB (DTLB)。ITLB 和 DTLB 也称为微 TLB，是由硬件进行管理的，软件不可访问。微 TLB 包含 JTLB 的子集。



1. 只有 4KEc 核有 JTLB。

2. 只有 4KEc 核有 ITLB/DTLB，4KEm 和 4KEp 核中有 FM。

图 13-16 MIPS 4KE 处理器内存管理单元 (经 MIPS 许可使用)

进行地址转换时，首先访问相应的指令（I）或数据（D）微 TLB。如果没有找到匹配的条目，将通过 JTLB 进行地址转换，并用转换后的地址重填微 TLB。如果在 JTLB 中没有发现条目，则会发出异常。为了减小 TLB 失效开销，读取数据时，在查找 DTLB 的同时也会查找 JTLB。这样当发生 DTLB 失效时，将会产生一个时钟周期的停顿，对于 ITLB 失效时，会产生两个时钟周期的停顿。

### 13.2.5 cache 控制器

数据 cache 控制器和指令 cache 控制器支持各种 cache 大小、组织方式和组相联方式（见图 13-17）。例如，数据 cache 可以是大小为 2K 字节、两路组相连，而指令 cache 可以是大小为 8K 字节、四路组相连。

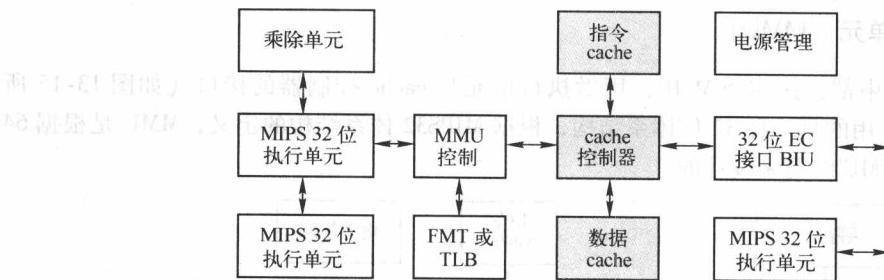


图 13-17 MIPS 4KE cache 控制器 (经 MIPS 许可使用)

指令 cache 和数据 cache 都有单独的 cache 控制器。每一个 cache 控制器都包含并控制着一个 1 行的填充缓冲区。除了要存放等待写入 cache 的数据之外，填充缓冲区可以与 cache 并行被访问，数据可以被旁路回到内核。关于 cache 内存设计的详细讨论，读者可以访问网站 <http://lwn.net/Articles/252125/> 进行详细了解。

### 13.2.6 总线接口单元 (BIU)

BIU 如图 13-18 所示，其控制着外部接口信号。此外，还有一个 32 位字节的折叠式写缓冲区。这个缓冲区在将数据发送到外部接口之前，为写操作提供支持。

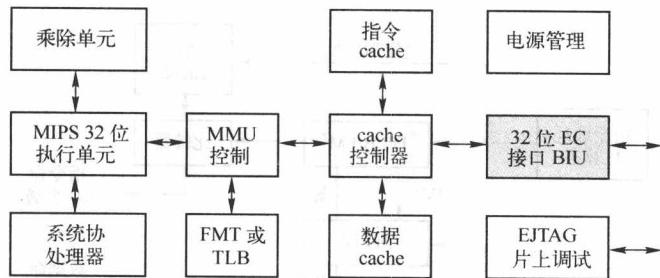


图 13-18 MIPS 4KE 总线接口单元 (经 MIPS 许可使用)

由于所有核的数据 cache 都采用写直达的 cache 策略，写缓冲可以极大地降低外部接口的写操作次数，并且，如果在很短的时间内进行多次写操作，还能够减小内核的停顿。

写缓冲由 2 个 16 字节的缓冲区组成。每一个缓冲区中的数据来自单个 16 字节对准的内存块。一个缓冲区保存的是当前正在相外部接口间传输的数据，而另外一个缓冲区中保存的是来自内核的数据。

### 13.2.7 电源管理

处理器核中设计了几种电源管理模式，包括低功耗设计、有效电源管理和掉电模式操作，如图 13-19 所示。内核支持 WAIT 指令，该指令用来通知其他设备执行操作和时钟将被挂起，这样可以在空闲期间降低系统功耗。

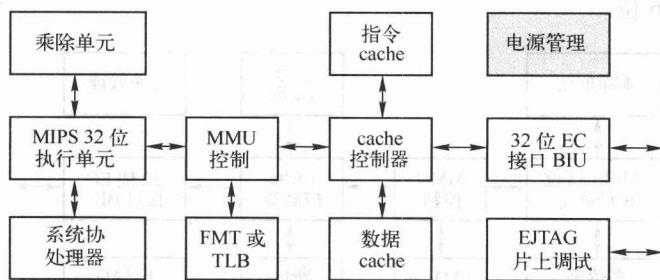


图 13-19 MIPS 4KE 电源管理（经 MIPS 许可使用）

内核提供了如下两种机制，用于支持系统级低功耗。

- 寄存器控制的电源管理。
- 指令控制的电源管理。

在寄存器控制的电源管理模式中，内核在 CPO 状态寄存器中设置了 3 位，用于软件控制的电源管理功能，即使是内核处于掉电模式，也可以对中断提供服务。在指令控制模式下，在掉电模式下执行 WAIT 指令可以触发低功耗模式。

### 13.2.8 指令 cache

指令 cache 是一种可选的片上内存阵列，最多可以是 64K 字节，如图 13-20 所示。cache 索引是虚拟的，标示是物理的，可以将虚地址转换和 cache 访问并行进行，而不必要等待物理地址转换结束。标示是 22 位的物理地址，一个有效位和一个锁位。还有一个单独的标示阵列，用于保存在最近使用 (least recently used, LRU) 替换模式中使用的数据。根据不同的相联度，LRU 阵列范围为 0 ~ 6 位。

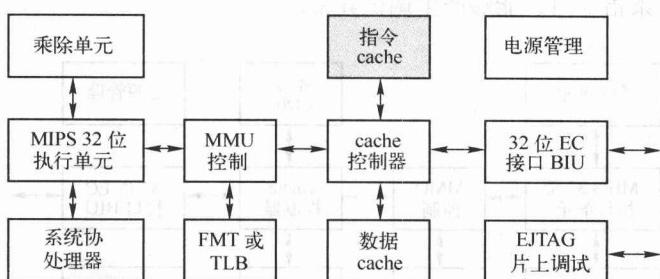


图 13-20 MIPS 4KE 指令 cache（经 MIPS 许可使用）

所有核都支持指令 cache 锁功能。cache 锁在“per-line”的基础上，将关键代码锁到 cache 中，使系统设计者获得最大的系统 cache 效率。一般在所有的指令 cache 条目中都有 cache 锁功能，可以通过执行 CACHE 指令将条目锁为锁定和解锁（将锁位设置或清除）。LRU 阵列必须是按位可写的，标示和数据阵列只需要字可写。

### 13.2.9 数据 cache

数据 cache 是一种可选的片上内存阵列，最多可以有 64K 字节，如图 13-21 所示。cache 索引是虚拟的，标示是物理的，可以将虚实地址转换和 cache 访问并行进行。标示是 22 位的物理地址，一个有效位和一个锁位，还有一个单独的标示阵列，用于保存 LRU 位。根据不同的相联度，LRU 阵列范围为 0~6 位。

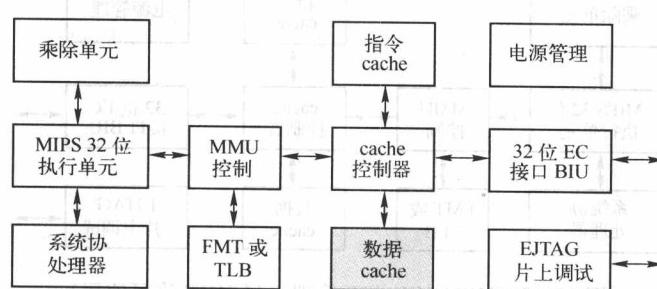


图 13-21 MIPS 4KE 数据 cache (经 MIPS 许可使用)

除了指令 cache 锁功能之外，所有的核也支持与指令 cache 相同的数据 cache 锁机制，可以以行为单位，将关键的数据段锁定到 cache 中。发生 cache 失效后，被锁定的内容不能替换；但是当发生写匹配时，可以被更新。

在所有的数据 cache 条目中都有 cache 锁功能，可以执行 CACHE 指令，将某个条目设为锁定或解锁状态。物理数 cache 内存必须是字节可写的，以支持半字存操作。LRU/污染位阵列必须是位可写的。

### 13.2.10 EJTAG 控制器

所有核中都设计了基本的 EJTAG 作为内核的一部分，以支持调试模式、运行控制、单步控制和软件断点指令，如图 13-22 所示。使用这些功能可以对用户和内核代码进行基本的软件调试，可选的 EJTAG 功能包括硬件断点。4KE 处理器核具有 4 个指令断点和 2 个数据断点，或者 2 个指令断点和 1 个数据断点，或者没有断点。用户可以对硬件指令断点进行适当的配置，使得在虚拟地址空间执行一条指令时，能够产生调试异常。

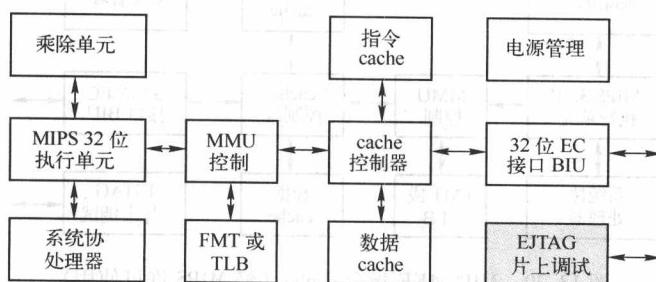


图 13-22 MIPS 4KE EJTAG 控制器 (经 MIPS 许可使用)

这些断点并不像软件指令断点一样仅限于 RAM 中的代码。用户可以对数据断点进行适当的配置，可以在进行数据处理时，避免产生调试异常。数据处理可以具有虚拟地址、数值、大小和 load/store 等处理类型。

在内核中还可以设计另外一个 TAP，可以通过专用端口，在 EJTAG 探测器和 CPU 之间进行通信。这样可以在应用程序中没有调试代码的情况下进行调试，并且可以用于将应用代码下载到系统中。另外一个可选的模块是 EJTAG 跟踪模块，可以进行实时跟踪。跟踪信息可以保存在片上跟踪内存中，也可以保存在片外跟踪探测器中。程序流的跟踪非常灵活，可以包括指令程序计数器，也可以包括数据地址和数据值。跟踪功能提出了一种功能强大的软件调试机制。

### 13.2.11 系统协处理器

可选的系统协处理器（CP2）接口为协处理器提供了一个全功能接口，如图 13-23 所示。它完全支持所有 MIPS32 COP2 指令，但是 64 位 load/store 指令（LED2、SDC2）除外。

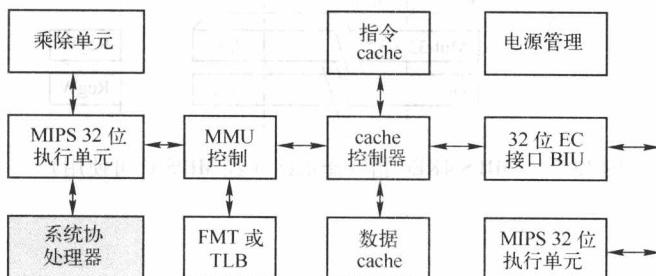


图 13-23 MIPS 4KE 系统协处理器 (经 MIPS 许可使用)

### 13.2.12 用户自定义指令 (UDI)

这个可选的模块（如果有）支持用户自定义的指令。这些指令必须在 MIPS 4KE 内核构造时进行定义。在操作码映射中共有 16 条指令可以用作 UDI，每一条指令可以是单周期的，也可以是多周期的。一条 UDI 指令可以对任何一个或两个通用寄存器，或者包含在指令中的立即数进行处理，并且指令的运算结果必须写回通用寄存器中。

### 13.2.13 指令流水线

MIPS32 4KE 处理器核中实现了 5 站流水线。该流水线可以在提高效率的同时，降低处理器的复杂性，减小成本和功耗。

流水线包括的 5 站有：

- 取指 (I 站)；
- 执行 (E 站)；
- 访存 (M 站)；
- 对齐 (A 站)；
- 写回 (W 站)。

3 个内核中都采用了“旁路”机制，可以将一次操作的结果直接发送给需要的指令，而不需要将结果写回寄存器后再读取。图 13-24 给出了 4KEc 处理器中每一个流水站所执行的操作。

#### 1. 取指

取指 (I) 站中：

- 指令从指令 cache 中取出；
- TLB 执行虚实地址转换（只有 4KEc 核有）。

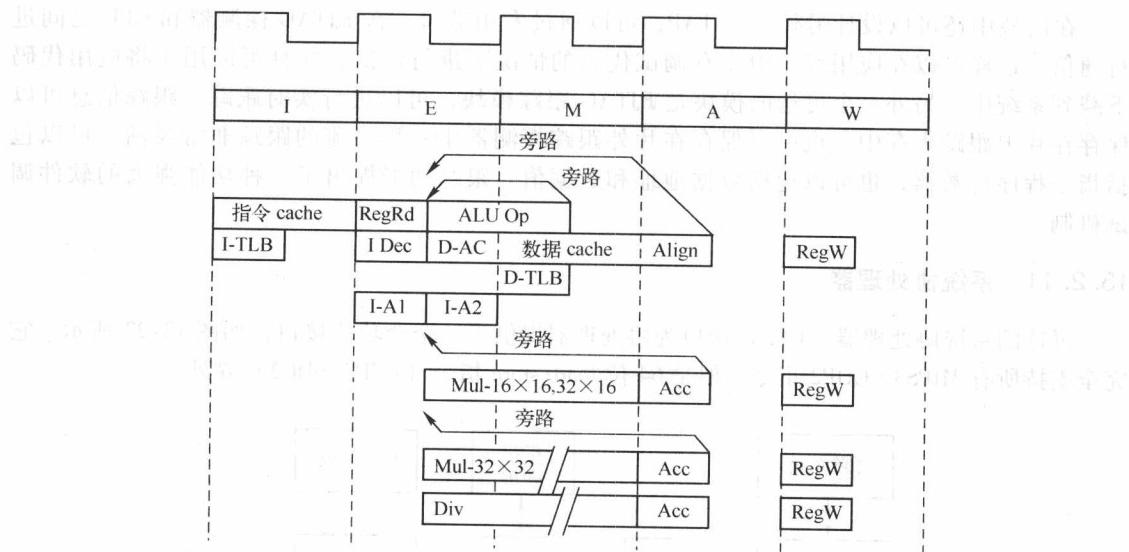


图 13-24 MIPS 4KEc 指令流水站（经 MIPS 许可使用）

## 2. 执行

执行 (E) 站中：

- 从寄存器文件中取出操作数；
- 来自 M 和 A 站的操作数旁路到本站；
- 对于 R-R 指令，算术逻辑单元 (ALU) 开始进行算数或逻辑操作；
- 对于 load 和 store 指令，ALU 计算数据的虚拟地址；
- 对于分支指令，ALU 确定分支条件是否为帧，并计算虚拟的分支目标地址；
- 指令逻辑选定指令地址；
- 所有的乘法和除法操作在本站开始执行。

## 3. 访存

访存 (M) 站中：

- 算术或逻辑 ALU 操作结束；
- 对于 load 和 store 指令，执行数据 cache 取和数据虚实地址转换操作；
- 执行数据 TLB (仅 4KEc 核有) 和数据 cache 查找操作，并确定是命中还是失效；
- $16 \times 16$  位或  $32 \times 16$  阵列操作完成，并在本站停止一个时钟周期，完成本站的进位传递加法 (对于 4KEc 和 4KEm 核)；
- $32 \times 32$  的 MUL 操作在本站停止 2 个时钟周期，以完成阵列的第二个周期，以及本站的进位传递加法 (于 4KEc 和 4KEm 核)；
- 在 MDU 中进行乘法和除法计算。如果在 IU 将前面的指令移动之前完成了计算，则 MDU 会将结果保存在暂存器中，直到指令单元将指令移动到对齐站为止。

## 4. 对齐

对齐 (A) 站中：

- 定位器将数据和它的字便捷进行对准；
- MUL 操作产生了等待写回的结果。实际的寄存器写回操作在写回站执行 (所有的 4KE 核)；
- 从这一站开始，从 MDU 装载数据或结果都可以在执行段获得，以进行旁路。

### 5. 写回

写回 (W) 站中：如果命中，数据将从 cache 行写回总线。如果未命中，忽略此站。

- 对于 R-R 或 load 指令，结果被写回寄存器文件中。

### 13.2.14 指令 cache 失效

当访问指令 cache 时，会对指令地址进行翻译，以确定所需要的指令是否在 cache 中。当所需要的指令地址不在指令 cache 中时，就发生了指令 cache 失效。当在取指站遇到 cache 失效时，内核将转换到执行站。流水线会在执行站停止，直到失效解决为止。总线接口单元必须从多个地址源中选择地址。如果地址总线忙，请求将会一直停留在仲裁站，直到总线空闲为止。内核将选择的地址发送到总线上，数据返回需要经历的时钟周期数也会由包含数据的阵列确定。一旦数据返回内核，所需要的数据将会写回指令寄存器中，以备立即使用。旁路机制可以使内核在数据到达时马上使用该数据，而不必等待将整个 cache 行都写入到指令 cache 中后再读取需要的数据。

图 13-25 给出了发出指令失效时的时序图。

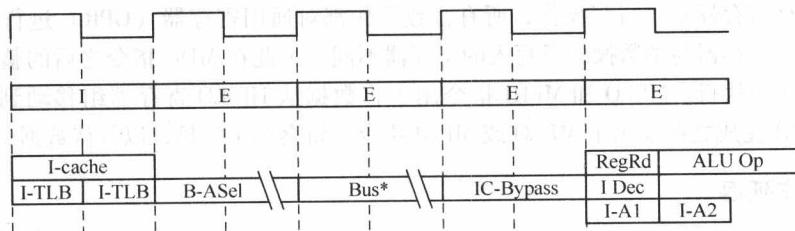
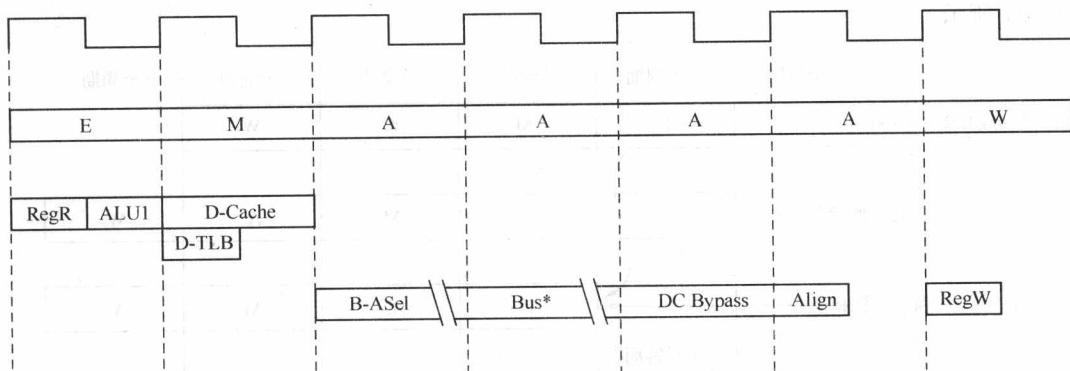


图 13-25 MIPS 4KE 指令 cache 失效时序 (经 MIPS 许可使用)

### 13.2.15 数据 cache 失效

访问数据 cache 时，会对数据地址进行转换，以确定需要的数据是否在 cache 中。当所需要的指令地址不在数据 cache 中时，就发生了数据 cache 失效。当在访存站 (D-TLB) 发现数据 cache 失效时，内核将转换到对齐站，流水线会停止在对齐站，直到失效被解决 (需要的数据读回)。图 13-26 给出了当发生数据 cache 失效时的时序图。



注：\*包括地址和数据使用总线的所有周期。

图 13-26 MIPS 4KE Load/Store cache 失效时序 (经 MIPS 许可使用)

总线接口单元在多个请求之间进行仲裁，并选择要发送到总线上的正确地址。内核将所选中的地址发送到总线上。数据返回需要的时间由包含数据的阵列决定。一旦数据被返回到内核，在定向到执行单元之前，需要的数据字将通过定位器被发送出来。旁路机制可以使内核在数据到达时马上使用该数据，而不必等待将整个的 cache 行都写入到指令 cache 中后再读取需要的数据。

### 13.2.16 乘法/除法操作

3 个内核中都设计了标准的 MIPS II 乘法和除法指令。此外，为了增强性能，还增加了许多新的指令。指标乘法指令，即 MUL 表明，乘法操作的结果将存放在通用寄存器文件中，而不是存储在 HI/LO 寄存器对中。为了避免使用 MFLO 指令，该指令在对 LO 寄存器操作时使用，并且支持多目的寄存器，对于乘法操作比较多的应用，可以提高吞吐量。

有 4 条指令用于执行乘加和乘减操作，分别是乘加（MADD）、乘加无符号（MADDU）、乘减（MSUB）、乘减无符号（MSUBU）。MADD/MADDU 指令将两个数据相乘，然后将结果与 HI 和 LO 寄存器中的值相加。类似地，MSUB/MSUBU 指令将两个操作数相乘，并用 HI 和 LO 寄存器的值减去结果。

MADD/MADDU 和 MSUB/MSUBU 操作通常在 DSP 算法中使用。所有乘法操作（除 MUL 指令之外）都对 HI/LO 寄存器对进行写操作，所有整数操作都对通用寄存器（GPR）进行写操作。由于 MDU 操作写入的寄存器与整数操作所写入的寄存器不同，因此在 MDU 指令之后的整数指令可以在 MDU 操作结束之前执行。MFLO 和 MFHI 指令用于将数据从 HI/LO 寄存器组移动到 GPR 文件中。如果在 MDU 操作完成之前发出了 MFLO 或 MFHI 指令，将会产生停顿，以等待数据。

### 13.2.17 分支延迟

流水线的分支延迟为 1 个时钟周期。这一个时钟周期的分支延迟是由于在流水线的执行段操作的分支决策逻辑产生的。这样在取指段要使用的分支目标地址在分支指令的两个周期后才能到达。在切换到分支目标地址之前，顺序的执行分支指令后的第一条指令，就插入了分支延迟槽。这样可以避免执行分支指令时，在流水线中插入气泡。地址计算和分支条件检查都在执行段进行。

流水线在延迟槽周期之后开始取分支路径或分支失败路径上的指令。当分支决策完成后，处理器继续取分支路径（如果分支成功）或者分支失败路径（如果分支失败）上的指令。分支延迟意味着跟随在分支指令之后的指令总是被执行，与分支的结构无关。如果在分支指令之后无法插入有用的指令，则编译器或汇编程序必须在延迟槽中插入 NOP 指令。分支延迟示意如图 13-27 所示。

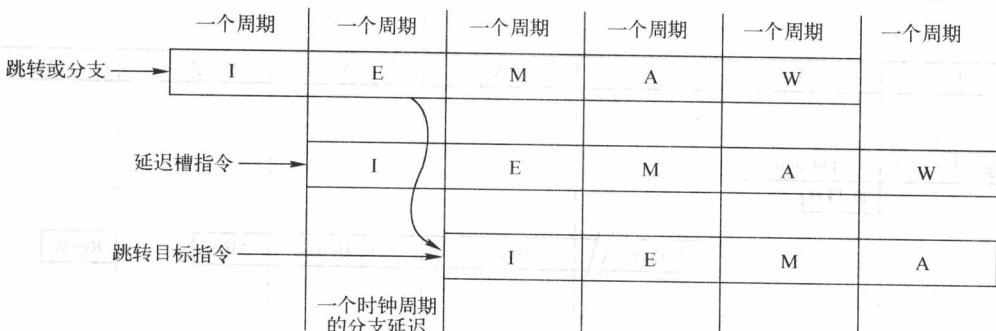


图 13-27 MIPS 4KE 流水线分支延迟示意图（经 MIPS 许可使用）

### 13.2.18 内存管理

MIPS 4KE 处理器核中设计了一个内存管理单元 (MMU)，是执行单元和 cache 控制器之间的接口。MIPS 4KEc 处理器核中包括一个 TLB，而 MIPS 4KEm 和 MIPS 4KEp 实现的是简单的直接映像风格的 MMU。

#### MMU 概述

在请求地址被送到 cache 控制器进行标示比较，或者送到接口单元进行外部存储器的访问之前，4KE 处理器核中的 MMU 可以将任何虚拟地址转换为物理地址，如图 13-28 所示。这种转换对于操作系统来说是非常有用的，可以帮助操作系统管理物理内存，实现在一个内存中容纳多个正在运行的任务。这些任务使用的虚拟地址可能相同，但是在物理内存中却处于不同的位置上（只有 4KEc 核有）。

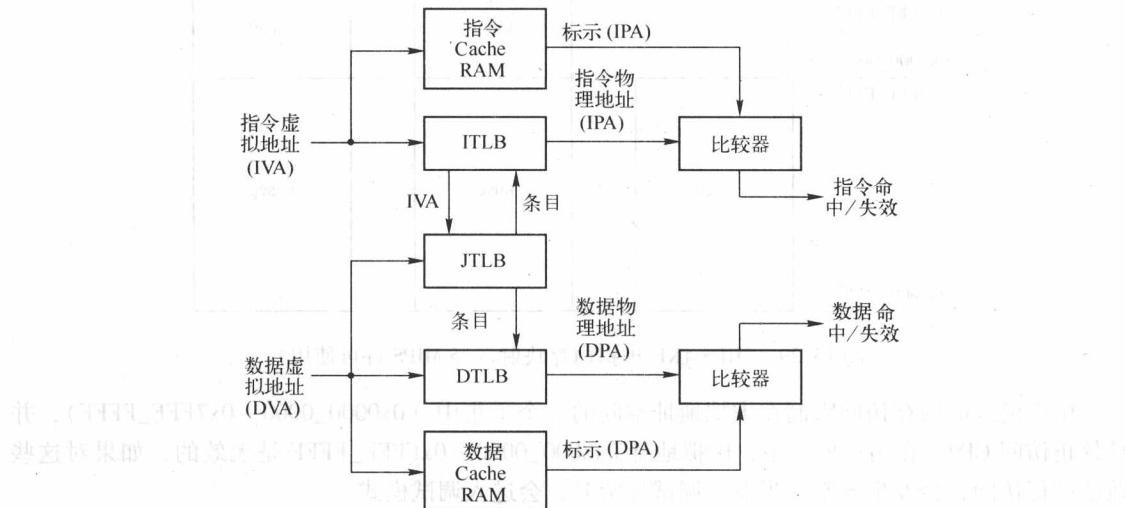


图 13-28 MIPS 4KE 进行 4KEc 核 cache 访问过程中的地址转换 (经 MIPS 许可使用)

MMU 的其他功能包括保护存储空间和定义 cache 协议。在 4KEc 处理器核中，MMU 是基于 TLB 的。

在 4KEm 和 4KEp 处理器核中，MMU 使用简单的算法，通过一种直接映射机制将虚拟地址转换为物理地址。这种转换对于不同的虚拟地址空间是不同的。

### 13.2.19 操作模式

操作模式如下 3 种：

- 用户模式；
- 内核模式；
- 调试模式。

对于应用程序来说，用户模式是最常用的。内核模式通常用于处理异常和优先的操作系统功能，包括 CPU 管理和 I/O 设备访问。调试模式用于软件调试，通常在软件开发工具中使用。MMU 执行的地址转换根据操作模式的不同而不同。

#### 1. 虚拟内存段

根据操作模式不同，虚拟内存段也不同。图 13-29 给出了由 32 位虚拟地址寻址的 4G 字节虚

拟内存空间，对应3种操作模式的分段情况。当复位和异常识别之后，处理器会进入内核模式。在内核模式中，软件可以访问整个地址空间，也可以访问所有CP0寄存器。

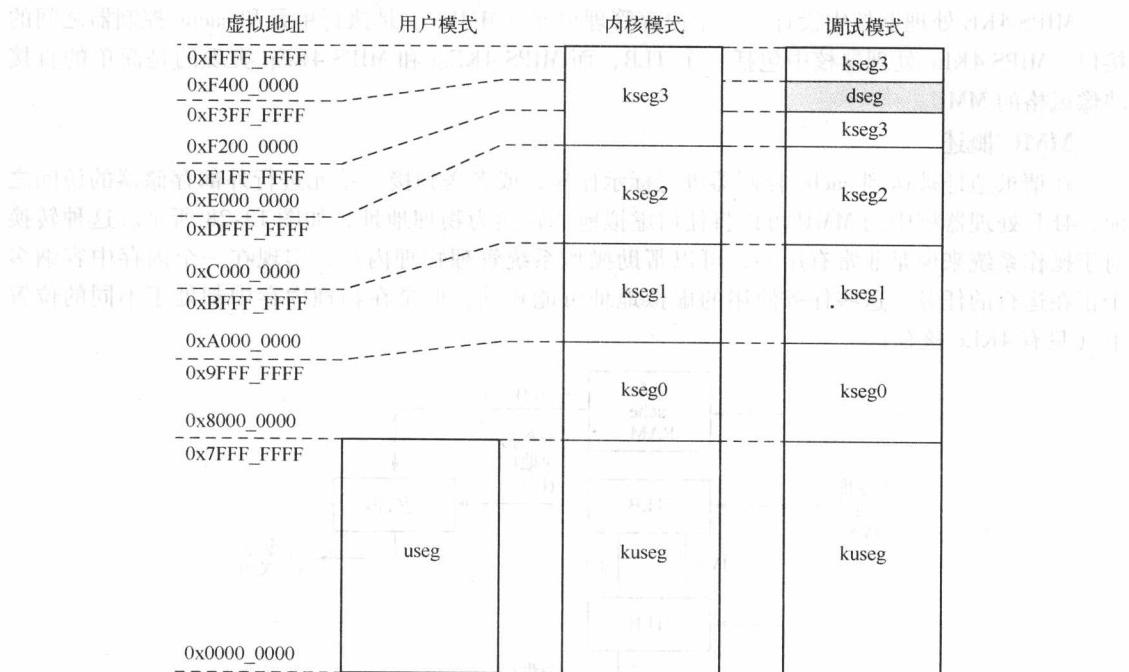


图 13-29 MIPS 4KE 虚拟内存映射 (经 MIPS 许可使用)

用户模式的内存访问限制在虚拟地址空间的一个子集中（0x0000\_0000 ~ 0x7FFF\_FFFF），并且禁止访问CP0。在用户模式中，虚拟地址0x8000\_0000 ~ 0xFFFF\_FFFF是无效的，如果对这些地址进行访问，会发生异常。当发生调试异常时，会进入调试模式。

在调试模式中，调试软件访问的地址空间和CP0寄存器与内核模式相同。此外，在调试模式下，处理器可以访问调试段dseg。这个区域与内核段kseg3有部分重合。调试模式下的dseg访问可以开启，也可以禁止。如果需要，允许调试模式访问整个kseg3。

## 2. 用户模式

在用户模式下，可以使用称为用户段(useg)的2G字节内存地址空间。图13-30给出了用户模式虚拟地址空间的位置。

## 3. 内核模式

处理器操作在内核模式，当调试寄存器中的DM位为0以及状态寄存器中包含一个或多个下列数据时，UM=0，ERL=1或EXL=1。

如果检测到非调试异常，处理器将进入内核模式。在异常处理子程序的最后，通常会执行一条异常返回(ERET)指令。ERET指令会跳转到异常PC，清除ERL。并且，如果ERL=0，表示要清除EXL，这样会使处理器返回用户模式。

内核模式的虚拟地址空间被分为区，由虚拟地址的高位进行区分，如图13-31所示。

## 4. 调试模式

关于映射和非映射的区域，调试模式的地址空间与内核模式的地址空间相同，但是kseg3除外。在kseg3中，调试段dseg在虚拟地址范围0xFF20\_0000 ~ 0xFF3F\_FFFF中共存。布局如图13-32所示。

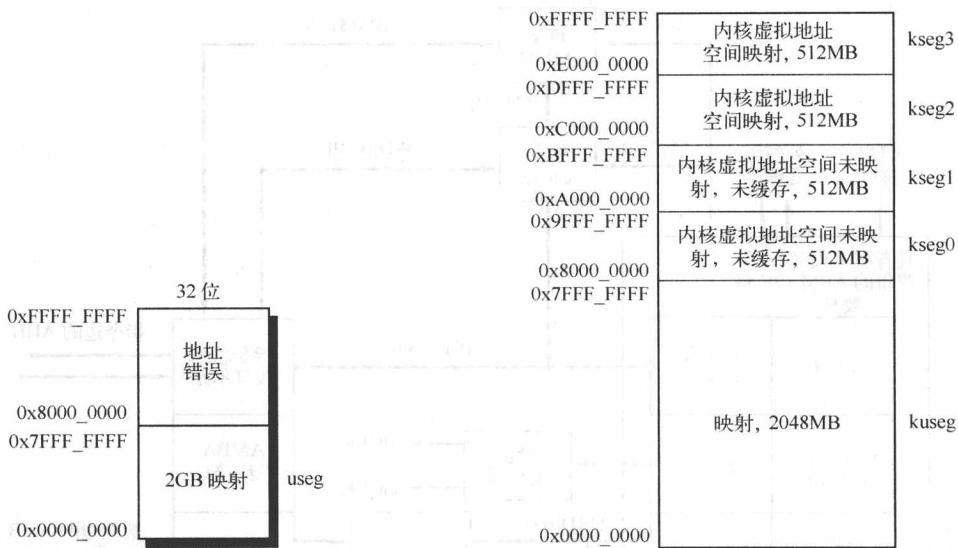


图 13-30 MIPS 4KE 用户模式虚拟地址空间（经 MIPS 许可使用）

图 13-31 MIPS 4KE 内核模式虚拟地址空间（经 MIPS 许可使用）

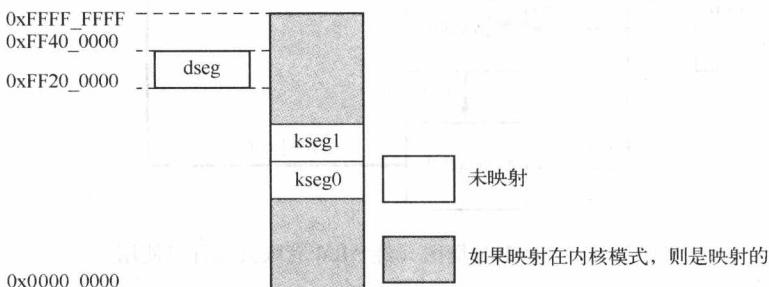


图 13-32 MIPS 4KE 调试模式虚拟地址空间（经 MIPS 许可使用）

### 13.3 ARM1022E 处理器概述

ARM1022E 处理器中集成了 ARM10E 整数核，该核实现的是 ARMv5TE 体系结构，如图 13-33 所示。它是一个高性能、低功耗，并且设计有 cache 的处理器，具有完整的虚拟内存功能；专门用于高端的嵌入式应用以及复杂的操作系统，例如 JavaOS、Linux 和 Microsoft WindowsCE；支持 ARM 和 Thumb 指令集，并包括 EmbeddedICE-RT 逻辑和 JTAG 软件调试功能。

ARM1022E 处理器由下列部件组成：

- ARM10E 整数核；
- Load/Store 单元；
- 指令预取单元；
- 整数单元；
- Embedded ICE-RT 逻辑，用于基于 JTAG 的调试；
- 外部协处理器接口和协处理器 CP14 和 CP5；
- 内存管理单元 (MMU)；
- 指令和数据 cache；

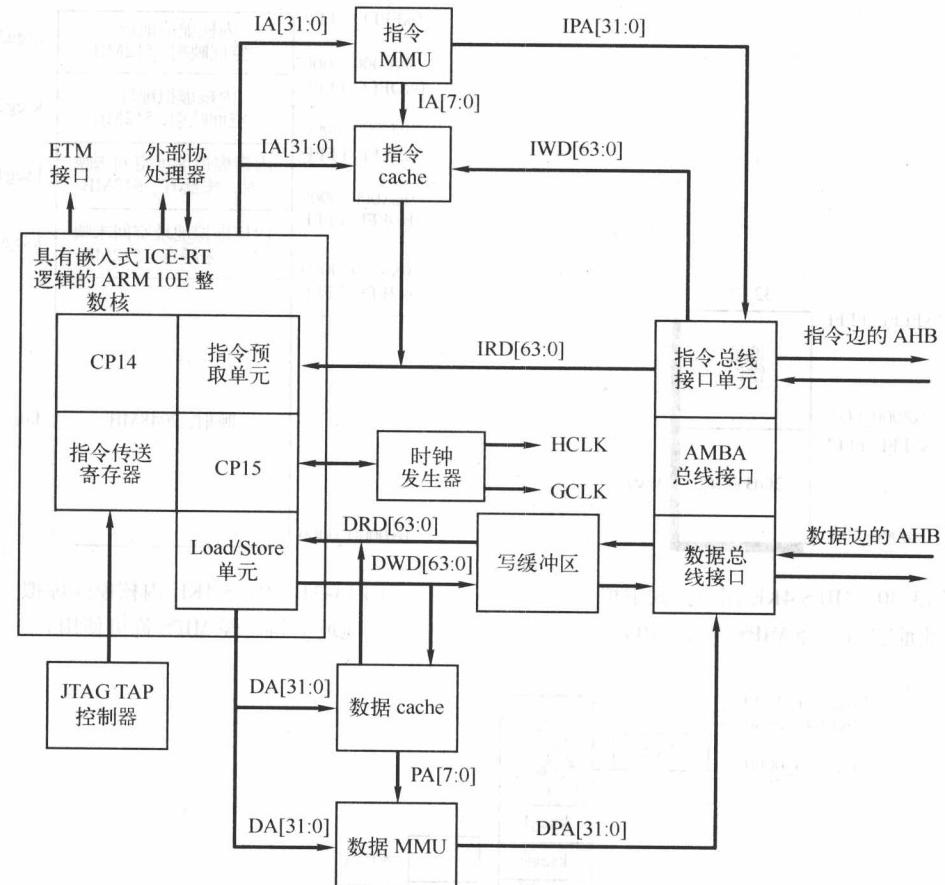


图 13-33 ARM10E 框图 (经 ARM 有限公司许可使用)

- 写回物理地址 (physical address, PA) TAG RAM;
- 写缓冲和失效下命中 (hit-under-miss, HUM) 缓冲区;
- 先进的微总线结构 (advanced micro bus architecture, AMBA) 高性能总线接口;
- 嵌入式跟踪宏单元 (embedded trace macrocell, ETM) 接口。

ARM1022E 处理器的特点有：

- 流水线为 6 站；
- 分支预测功能，以支持零延迟的分支展开；
- 32KB 一级 cache，包含 16KB 的指令 cache 和 1KB 的数据 cache；
- 整数核和 cache、写缓冲区之间具有完全的 64 位接口，在指令和数据侧以及协处理器都有总线接口单元；
- 在指令和数据侧通过独立的 64 位 AHB 接口支持多层 AHB；
- 在运行 load-store 多条指令的情况下，并行执行数据处理；
- HUM 缓冲区，在发生装载失效的情况下，支持装载命中；
- 非阻塞的 cache，在发生 load 失效的情况下，支持数据处理指令的执行；
- 附加的寄存器读写端口，在一个周期之内，可以支持最多 4 个寄存器的读操作和 3 个寄存器的写操作；

- 增强的电源管理支持；
- 高级的调试支持。

### 13.3.1 处理器组成

本节将介绍 ARM1022E 处理器的主要组成模块，这些模块的描述将在其余各节中介绍。ARM1022E 处理器的主要模块包括：

- 整数核；
- 内存管理单元；
- 指令和数据 cache；
- cache 节电功能；
- 分支预测和指令预取单元；
- AMBA 接口；
- 协处理器接口；
- 调试；
- 指令周期统一和互锁；
- 可测性设计特性；
- 电源管理；
- 时钟供给和 PLL。

#### 整数单元

ARM1022E 处理器是围绕 ARM10E 整数核构建的，具有 ARMv5TE 体系结构，可以运行 32 位的 ARM 指令集和 16 位的压缩 Thumb 指令集（见图 13-34）。这样可以在高性能和代码尺寸之间进行很好的折中，在使用 8 位、6 位、32 位内存时获得最高的性能。处理器包括 Embedded ICE-RT 逻辑，用于 JTAG 软件调试，并由 Multi-ICE JTAG 调试接口提供支持。

### 13.3.2 寄存器

ARM1022E 处理器中共有 37 个 32 位的寄存器，如图 13-35 所示。

- 16 个通用寄存器。
- 1 个状态寄存器。
- 15 个分组的寄存器（与特定模式相关的通用寄存器）。
- 5 个分组的寄存器（与特定模式相关的状态寄存器）。

这些寄存器都可以同时进行访问。处理器状态和处理器操作模式确定了哪些是程序员可以使用的寄存器。

### 13.3.3 整数核

整数核（见图 13-36）将流水线中各个站的操作重叠进行，将每条指令执行时可以获得的时钟频率最大化。因为内核中含有多个执行单元，可以将多条指令置于同一个流水站，可以同时对一些指令进行处理。因此，它的最高吞吐量为每周期一条指令。

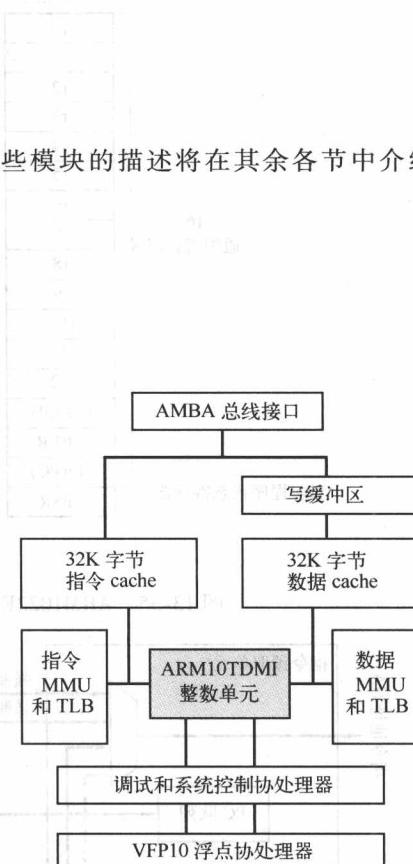


图 13-34 ARM10E 整数单元框图  
(经 ARM 有限公司许可使用)

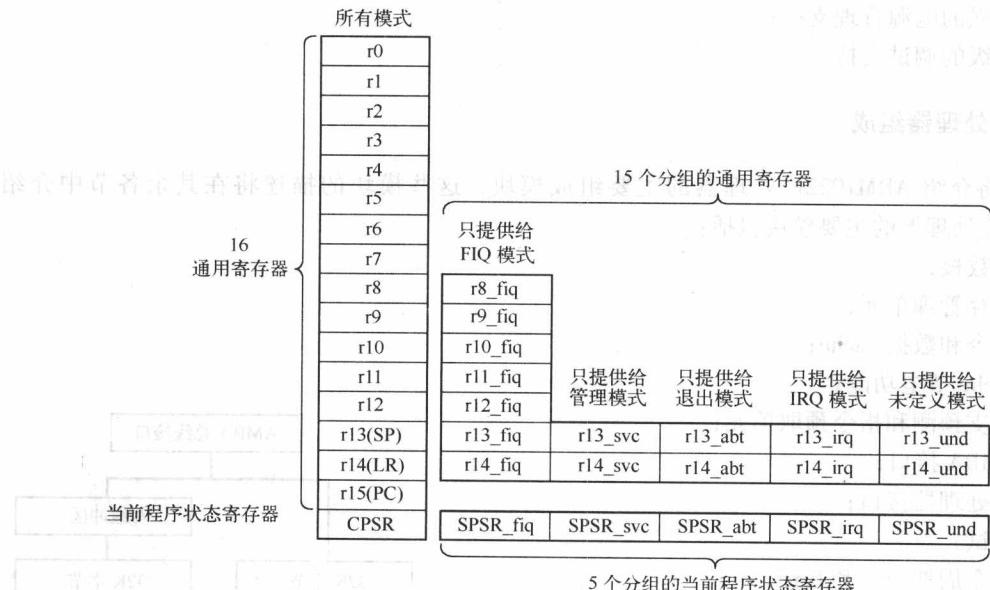


图 13-35 ARM1022E 寄存器集合 (经 ARM 有限公司许可使用)

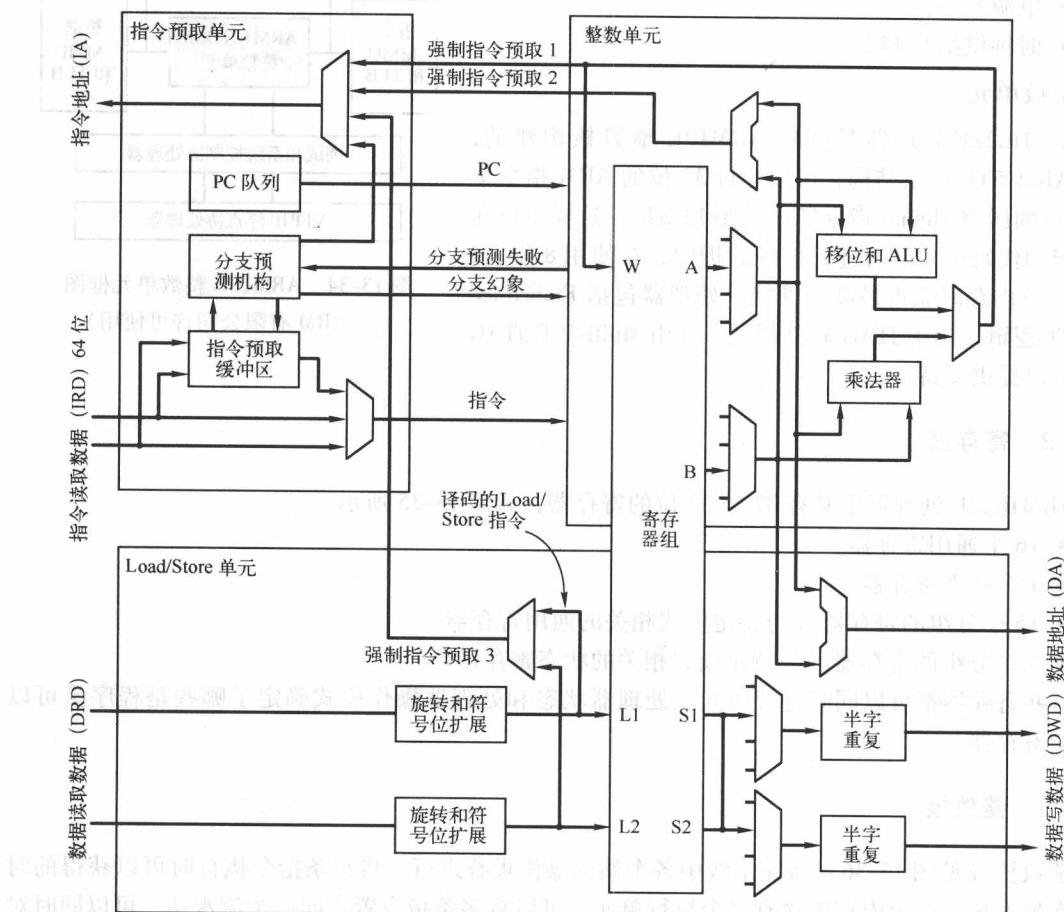


图 13-36 ARM1022E 整数核框图 (经 ARM 有限公司许可使用)

### 13.3.4 整数核流水线

ARM10 整数流水线共有 6 站，可以极大地提高指令吞吐量，如图 13-37 所示。流水线具有如下功能：

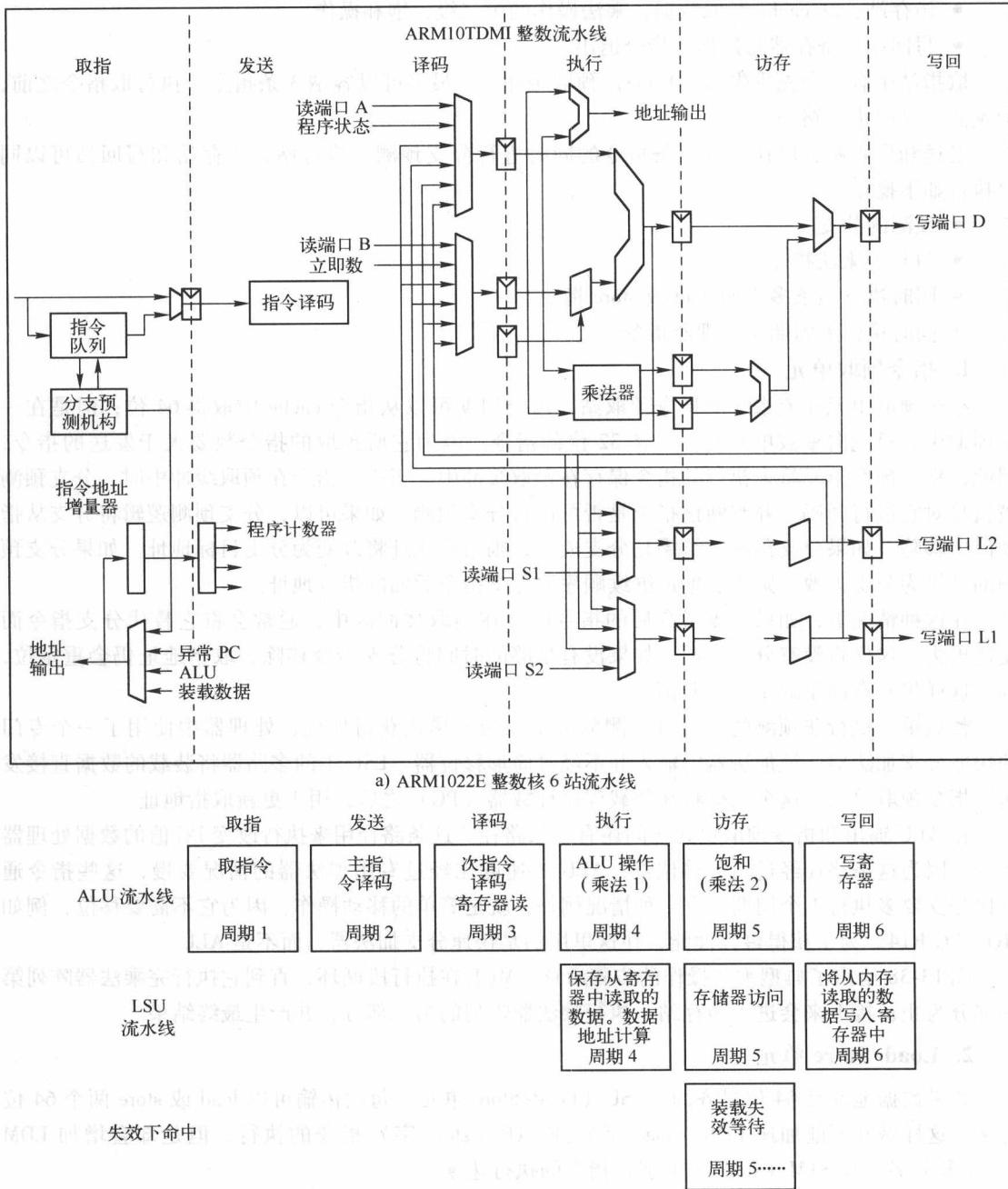


图 13-37 (经 ARM 有限公司许可使用)

- 取指站：从 cache 中获得指令，对取出的分支指令进行分支预测。
- 发送站：进行指令的初始译码。

- 译码站：对指令进行最终译码，为算术逻辑单元（ALU）的操作读取寄存器、定向和初始化的互锁裁决。
- 执行站：计算访存地址，数据处理移位，移位和饱和操作，ALU 操作，乘法操作的第一级，标志位设置，条件码检查，分支预测检测，将读取的数据寄存器内容保存到内存。
- 访存站：从 cache 获取数据，乘法操作的第二级，饱和操作。
- 写回站：寄存器写操作，指令退出。

取指站中有一个先进先出（FIFO）预取缓冲区，最多可以容纳 3 条指令。执行取指令之前，会预测一个取指的路径。

发送和译码站可以在执行一条指令的同时进行分支预测。执行站、访存站和写回站可以同时执行如下操作。

- 预测的分支。
- ALU 或乘法指令。
- 同时执行多条多周期 load 或 store 指令。
- 同时执行多周期协处理器指令。

## 1. 指令预取单元

指令预取单元在流水线的指令预取站，每个周期可以从指令 cache 中取得 64 位；但是在在一个周期中，只能向整数单元发生一条 32 位的指令。因为它所预取的指令数要大于发送的指令，因此，指令预取单元将未执行的指令保存在预取缓冲中。当一条指令在预取缓冲中时，分支预测逻辑将对它进行译码，并判断该指令是否可进行分支预测。如果可以，分支预测逻辑将分支从指令流中移除。如果分支预测的结果是分支成功，则指令地址将改变为分支目标地址。如果分支预测的结果为分支失败，则指令地址继续顺序取分支指令后面的指令地址。

在这种情况下，如果分支指令后的指令已经在预取缓冲区中，通常会将它替代分支指令而发送出去，这样将没有分支开销。如果没有足够的时间将分支指令移除，取指地址仍会重定位，因为这样仍然有助于减小分支开销。

整数单元执行非预测的或不可预测的分支。为了尽快获得地址，处理器中使用了一个专门的快速分支加法器，该加法器的输入并不经过筒形移位器。LSU 中的多路器将装载的数据直接发送给指令预取单元。这个数据将在装载程序计数器（PC）之后，用于更新取指地址。

在 ALU 输出和指令预取单元之间还有一条路径。这条路径用来执行改变 PC 值的数据处理器指令。因为这条路径经过筒形移位器，ALU 的速度比经过专用加法器的情况要慢，这些指令通常比分支要多执行 1 个周期。有一种情况例外，就是简单的移动操作，因为它不需要移位，例如 MOV PC R14。为了获得最佳性能，在这里用的是快速分支加法器，而不是 ALU。

图 13-38 给出了典型乘法操作的步骤示意。MUL 在执行段循环，直到它执行完乘法器阵列第一部分为止；接下来会进入访存站，执行乘法器阵列的第二部分，并产生最终结果。

## 2. Load/Store 单元

如果数据地址是 64 位对齐的，LSU（Load/Store 单元）每次传输可以 load 或 store 两个 64 位的字。这样做并不能加速 LDR（load 字）或 STR（store 字）指令的执行，但是却会增加 LDM（load 多个字）和 STM（store 多个字）指令的执行速度。

如果数据地址不是 64 位对齐的，则访问时间将超过 2 个周期。如果 LDM 或 STM 指令不是 64 位对齐的，则第一次访问将只能传输一个 32 位的字。随后，每个周期可以传输两个字。单个字的 load 和 store 操作要与整数单元合作执行。多个字 load 和 store 操作中，第一个周期要与整数单元合作执行，但是 LSU 可以自治地完成正在进行的多个字 load 和 store 操作。

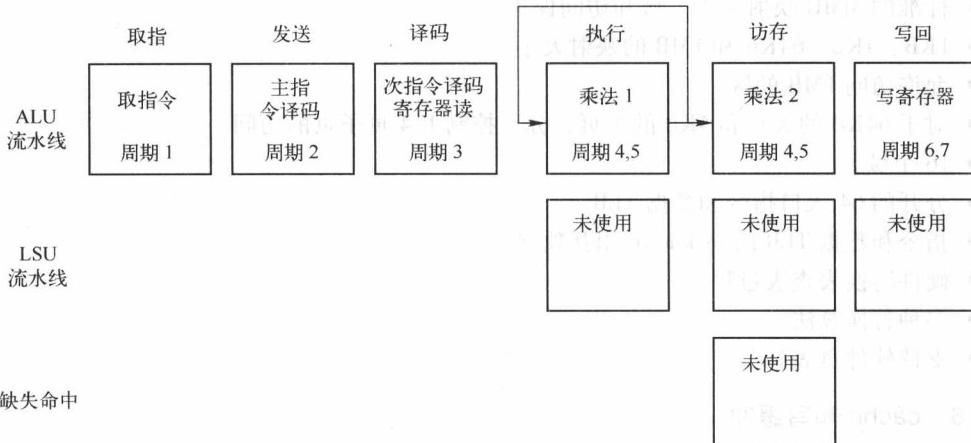


图 13-38 ARM1022E 典型乘法操作的流水线站（经 ARM 有限公司许可使用）

LSU 采用专用的加法器计算取数据的地址，该加法器与 ALU 中的加法器功能相同。如果需要，ALU 中的加法器计算基址寄存器的写回值。整数单元的 A 和 B 寄存器端口为两个加法器读取操作数。对于复杂的伸缩寄存器寻址模式，需要简形移位器的运行，ALU 必须计算数据地址，这个过程将再消耗一个周期。

LSU 具有两个专用的寄存器组读端口——S1 和 S2，以及两个专用的写端口——L1 和 L2。这些端口用于读取要保存的数据，以及写入要装载的数据。Load/Store 操作的步骤如图 13-39 所示。

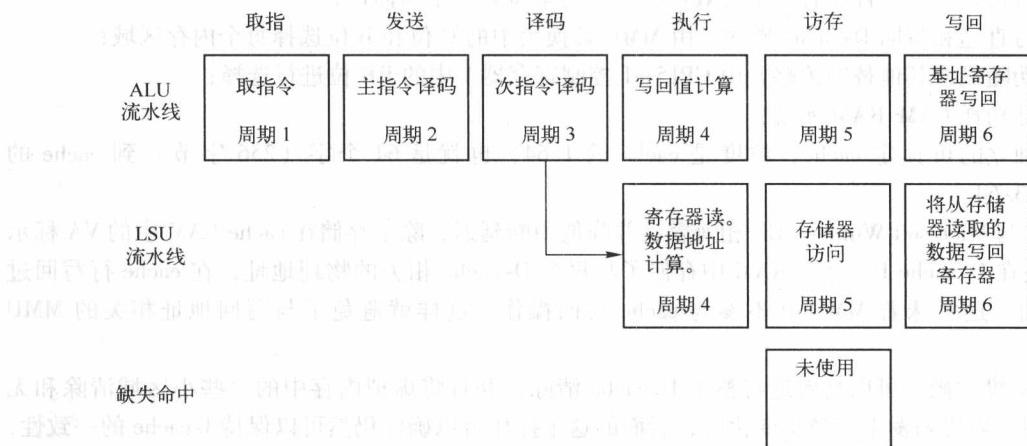


图 13-39 ARM1022E Load/Store 操作的流水线站（经 ARM 有限公司许可使用）

### 13.3.5 内存管理单元

MMU 具有分开的指令和数据 TLB。它与 StrongARM 和 ARM920T 的 ARM v4 体系结构 MMU 后兼容。MMU 包含一个 1KB 的小页映像，适用于较小的 RAM 和 ROM 封装的情况，嵌入式系统和类似于 Windows CE 的操作系统中会有很多小的映像目标。ARM1022E 处理器实现了快速上下文切换扩展 (fast context switching extension, FCSE) 和高向量扩展，在运行 Microsoft Windows CE 的时候会用到。

MMU 的主要功能有：

- 标准的 MMU 映射大小、域和访问保护。
- 1KB、4KB、64KB 和 1MB 的映射大小。
- 允许访问 1MB 的区。
- 对于 64KB 的大页和 4KB 的小页，分开控制 1/4 页子页的访问。
- 16 个域。
- 分开的 64 入口指令和数据 TLB。
- 指令和数据 TLB 的 lockdown 相互独立。
- 硬件转换表查表过程。
- 多种替换算法。
- 支持软件页表。

### 13.3.6 cache 和写缓冲

ARM 处理器包括：

- 一个指令 cache；
- 一个数据 cache；
- 一个写缓冲；
- 一个失效下命中 (HUM) 缓冲区。

16KB 的 I-cache 和 16KB 的 D-cache 具有以下特性：

- 8 个段，每个段有 64 行；
- 虚拟寻址的 64 路相连；
- 每行 8 位，并且还有一个有效位、一个污染位和一个写回位；
- 写直达和写回 D-cache 操作，由 MMU 转换表中的 C 位和 B 位选择每个内存区域；
- 伪随机或其他替换策略，由 CP15 c1 控制寄存器 1 中的 RR 位进行选择；
- 低功耗 CAM-RAM 实现；
- 独立的可锁定 cache，粒度是 cache 的 1/64，也就是 64 个字 (256 字节) 到 cache 的 63/64。

为了与 Microsoft Windows CE 相兼容，并降低中断延迟，除了存储在 cache CAM 中的 VA 标示以外，还在 D-cache PA 标示 RAM 中存储了与每个 D-cache 相关的物理地址，在 cache 行写回过程中使用。这意味着 MMU 并不参与 cache 写回操作，这样就避免了与写回地址相关的 MMU 失效。

cache 维护操作可以高效地将整个 D-cache 清除，并且将虚拟内存中的一些小区域清除和无效化处理。当代码发生了微小变化时，后面的这个操作可以确保仍然可以保持 I-cache 的一致性，例如自修改代码和对异常向量的改变。写缓冲可以容纳 8 个 64 位的数据包，每一个都有与之相关的地址单元。

### 13.3.7 总线接口

ARM10 处理器面向高级微控制器总线体系结构 (AMBA) 进行设计，并将 AMBA 高性能总线 (AMBA high-performance bus, AHB) 用作它与内存和外设的接口。

对于指令和数据，ARM10 处理器分别使用不同的 AHB 总线接口，包括：

- 指令总线接口单元 (instruction bus interface unit, IBIU)；
- 数据总线接口单元 (data bus interface unit, DBIU)。

当发生数据 cache 失效时，分离的总线接口可以使取指和执行并行进行，两个接口之间未共

享任何 AHB 信号。ARM10 AHB 接口总是有驱动的。如果没有任何总线主设备对总线进行控制，数设备会将“0”信号驱动到总线上，以避免总线竞争。

ARM10 处理器具有双向的输入、输出和控制信号。关于 AMBA 总线的详细描述，包括 AHB 总线和 AMBA 测试方法，请参考 AMBA 说明书。总线接口控制着内核时钟域和 AMBA 总线时钟域之间的所有数据传输和指令传输（见图 13-40）。任何来自指令预取单元或 LSU 的请求，如果需要传送到 ARM10 处理器外部，都会由总线接口进行控制，这种控制对于指令预取单元或 LSU 是透明的。

下述来自 cache 和 MMU 的请求都会驱动总线控制器：

- 由 MMU 产生的转换表查表过程；
- 不可高速缓冲的读取；
- 不可缓冲的写；
- 行填充；
- 经过缓冲的写；
- CP15 清空写缓冲和清除索引操作。

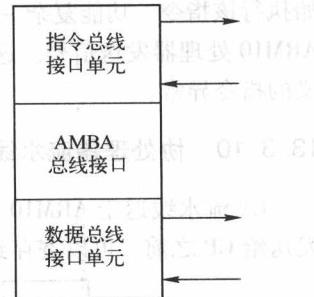


图 13-40 ARM1022E 总线接口单元  
(经 ARM 有限公司许可使用)

### 13.3.8 拓扑结构

总线接口包含两个完全分离的模块，对于 RISC 体系结构来说是非常典型的。指令总线接口单元 (IBIU) 处理所有取指和行填充操作，数据总线接口单元 (DBIU) 执行所有数据 load 和 store 操作。

当需要时，IBIU 和 DBIU 都可以执行转换表 walk 操作。图 13-41 给出了总线接口的结构，DBIU 在图的左边，有控制、读、写和地址通路；IBIU 在图的右边，只有一个读数据通路和一个地址数据通路，因为在指令段永远都不会发生写操作。

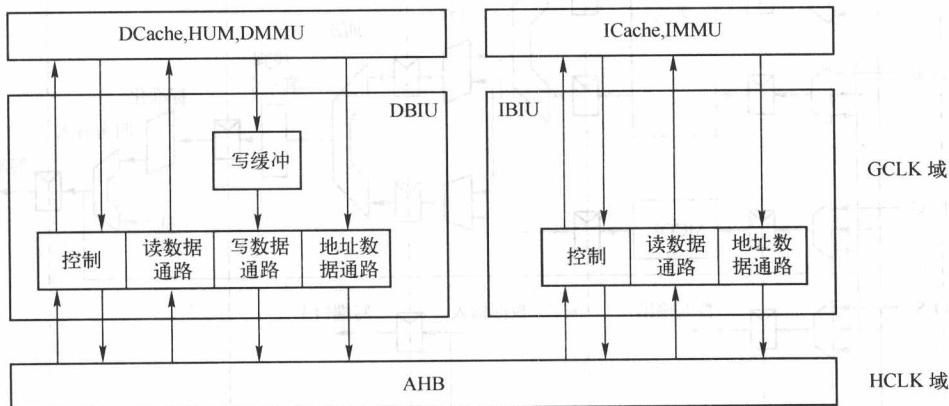


图 13-41 ARM1022E 总线接口框图 (经 ARM 有限公司许可使用)

在 HCLK 时钟域之间以及进一步向 AMBA 系统的其余部分传输数据或指令时，IBIU 和 DBIU 的层次非常相似。图中的箭头表示请求和数据或指令流的方向。

DBIU 和 IBIU 是相互独立的。在 ARM 处理器的数据和指令端之间，没有有效的方式进行通信，这样就很难支持自修改代码。

### 13.3.9 协处理器接口

协处理器接口可以将多个协处理器 (CP) 连接到 ARM10 处理器上。为了限制接口的连接数

量，每个 CP 都会跟踪 ARM10 流水线中正在执行的指令。为了使 CP 的性能最高，ARM10 处理器会尽早发送 CP 指令。也就是说，指令是看机会发送的，并且如果发生了异常或分支预测失败，还可以在流水线后期删除。因此，CP 必须能够在 ARM10 流水线的后面几站中删除指令。

功能简单的 CP 会一直跟踪 ARM10 流水线，直到确定给定的指令不会被删除。此时 CP 将开始执行该指令。功能复杂一些的 CP 重复利用指令提前发出这一点，在流水线的某些点上，CP 向 ARM10 处理器发送信号。这些信号表示 CP 需要更多的时间来执行指令，或者表示必须发出未定义的指令异常。

### 13.3.10 协处理器流水线

CP 流水线迟于 ARM10 流水线执行一个周期。这样可以确保 ARM10 处理器的流水线数据在发送给 CP 之前，可以寄存到寄存器中。图 13-42a 和图 13-42b 给出了 ARM10 和 CP 的流水线站。

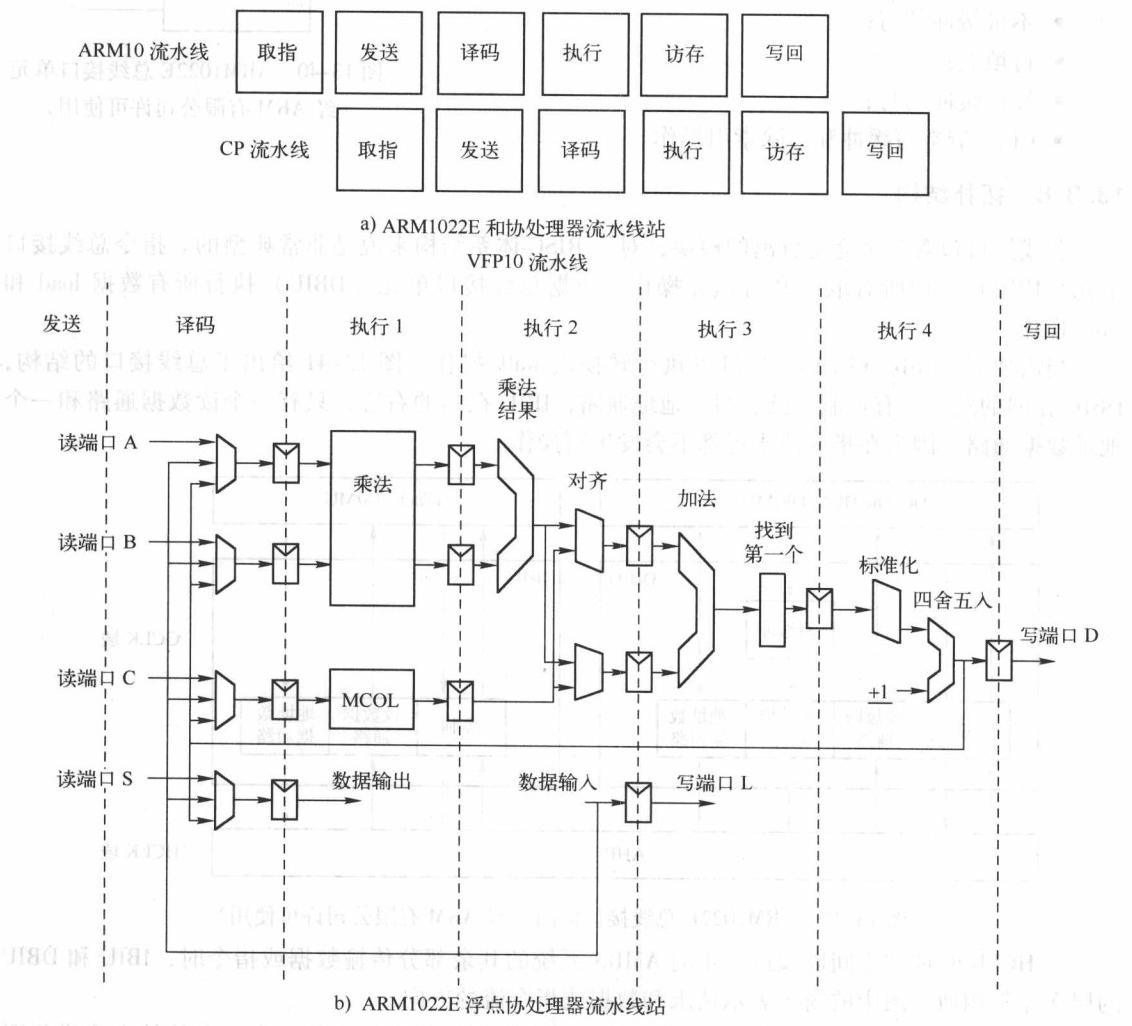


图 13-42 (经 ARM 有限公司许可使用)

### 13.3.11 调试单元

ARM1022E 处理器可以被嵌入到大型 SoC 设计中。嵌入式 ICE-RT 逻辑调试功能、AMBA 片

上系统总线和测试方法，都使得处理器嵌入到大型 IC 中后，可以提高使用效率。

调试单元给调试软件提供支持（见图 13-43）。

调试硬件与软件调试器程序相结合，可以用于调试：

- 应用软件；
- 操作系统；
- 基于 ARM10 的硬件系统。

调试单元可以完成如下操作：

- 停止程序执行；
- 检查和改变处理器以及协处理器的状态；
- 检查和改变内存和输入/输出外设的状态；
- 重新启动处理器核。

调试单元可以通过多种方式停止执行，最常用的方法是使执行过程在取指的断点或取数据的观察点停止。执行停止后，就会进入挂起模式或者监视器调试模式。

### 13.3.12 挂起模式

所有处理器都可以执行挂起，并且只能由连接到外部 JTAG 接口的硬件重启。此时可以通过 JTAG 接口检查并修改所有处理器寄存器、协处理器寄存器、内存和输入/输出端口。

这种模式可以干预程序的执行。在挂起模式中，可以对处理器进行调试，而不管它的内部状态如何。挂起模式需要外部硬件控制 JTAG 接口，软件调试器是用户与调试硬件之间的接口。

### 13.3.13 监视器调试模式

在监视器调试模式中，处理器停止当前程序的执行，开始执行调试退出处理程序。处理器状态的保存方式与所有 ARM 异常相同。

退出处理程序与调试器应用程序进行通信，可以获得处理器和协处理器的状态，还可以访问内存和输入/输出外设。监视器调试模式需要调试监视器程序作为调试硬件和软件调试器之间的接口。

### 13.3.14 时钟和 PLL

ARM1022E 处理器有 GCLK 和 HCLK 两个时钟输入。

该设计是完全静态的。当这两个时钟（见图 13-44）都处于停止状态时，处理器的内部状态将不确定。GCLK 驱动处理器的内部逻辑，HCLK 驱动总线接口，大多数输入和输出的时序都是根据 HCLK 确定的。

一般来说，GCLK 的频率要高于 HCLK，两个时钟必须具有固定的相位关系。HCLK 通常可以通过对 GCLK 分频来获得。

### 13.3.15 ETM 接口逻辑

一个可选的外部 ETM（外部跟踪模块），如图 13-45 所示，可以连接到 ARM1022E 处理器上，用于实时跟踪嵌入式系统中的指令和数据。处理器要提供相应的逻辑和接口，使用户可以使用 ETM10 来跟踪程序执行和数据传输。

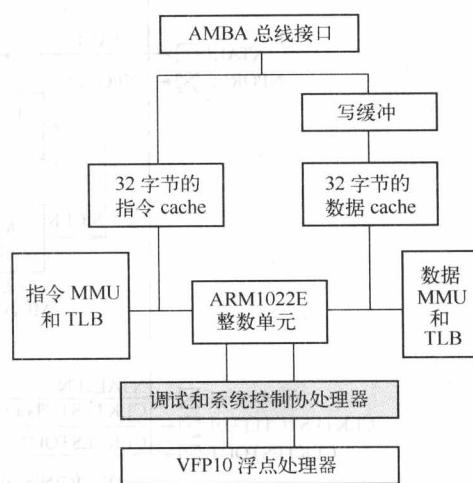


图 13-43 ARM1022E 调试控制器  
(经 ARM 有限公司许可使用)

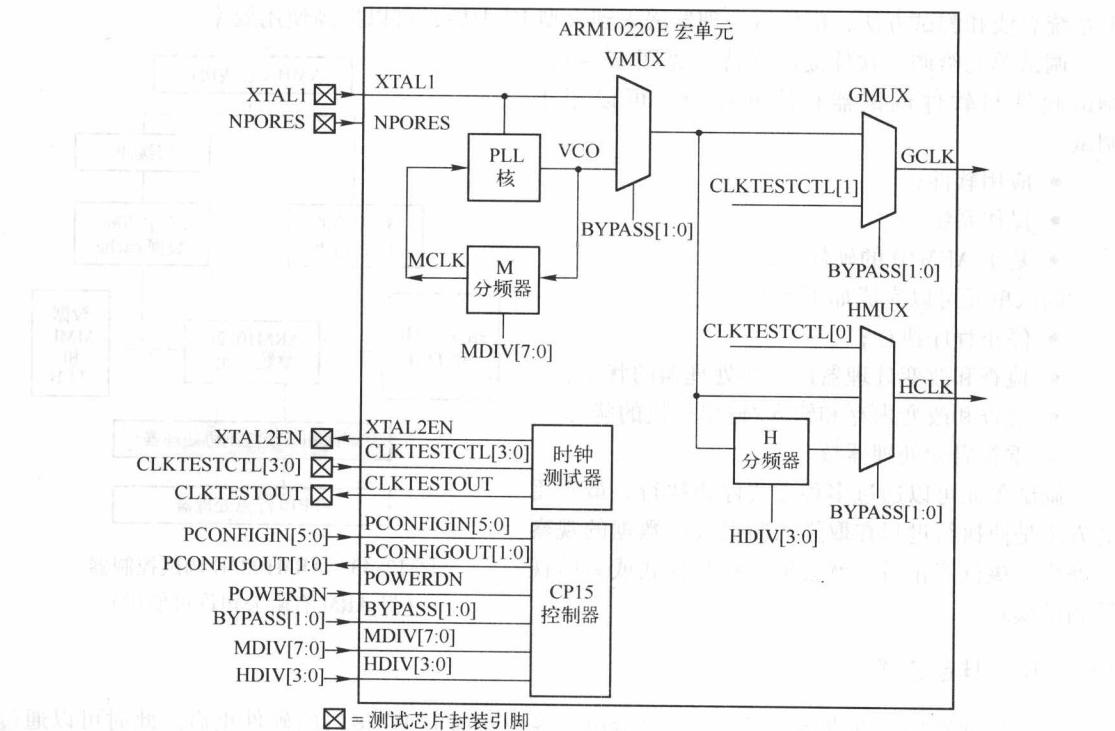


图 13-44 ARM1022E 时钟发生器（经 ARM 有限公司许可使用）

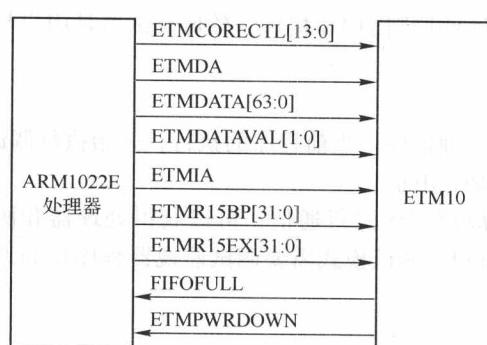


图 13-45 ARM1022E ETM 接口逻辑（经 ARM 有限公司许可使用）

### 13.3.16 工作状态

ARM1022E 处理器有两种工作模式，ARMv5TE 体系结构中的寄存器结构根据工作模式的不同而不同。

- ARM 状态——处理器执行 32 位、字对齐的 ARM 指令；
- Thumb 状态——处理器执行 16 位、半字对齐的 Thumb 指令。程序计数器（PC）使用位 1 对两个半字进行选择。

### 13.3.17 状态转换

通过如下方式，可以使处理器的操作状态在 ARM 和 Thumb 之间转换：

- 执行 BX 和 BLX 指令；
- 将 PC 装入 CP15 c1 控制寄存器 1 中清除的 L4 位。

处理器在 ARM 状态开始处理所有异常，如果在 Thumb 状态下发生了异常，处理器将转换到 ARM 状态；当从异常处理状态返回时，可以自动返回 Thumb 状态。

### 13.3.18 在异常处理中切换状态

异常处理器会将处理器切换到 Thumb 状态，但是最后必须返回 ARM 状态，否则异常处理器将无法正确结束。

### 13.3.19 工作模式

处理器共有如下 7 种工作模式：

- 用户：非特权模式，用于执行一般程序；
- 快速中断（fast interrupt, FIQ）：特权异常模式，用于处理快速中断；
- 中断（interrupt, IRQ）：特权异常模式，处理常规中断；
- 管理：操作系统功能的特权模式；
- 退出：特权异常模式，处理数据终止和指令预取终止；
- 系统：特权用户模式，用于操作系统功能；
- 未定义：特权异常模式，用于处理未定义的指令。

除了用户模式之外，其他模式统称为特权模式。特权模式用于异常服务或访问受保护的资源。

## 习题

1. 列出使用 SoC，而不使用 GPM 的 3 个关键原因。
2. RTL 设计有哪些限制？
3. 当从板级设计转换到 SoC 时，有哪些关键问题需要解决？
4. 列出使用巨大 RTL 设计的 3 个挑战。
5. 列出将 RTL 设计转换到 SoC 的 6 个挑战。
6. 列出 3 种不适合 SoC 的应用。
7. 为什么在 MIPS 4Ke 中有特殊的乘加指令？
8. 为什么在 MIPS 4Ke 中有可选模块？
9. MIPS 4Ke 指令和数据 cache 可选择的优点是什么？
10. 为什么要使用可选的 MIPS 4Ke EJTAG 特性？
11. 画图说明 MIPS 4Ke 的基本 5 站流水线。
12. 描述发生 MIPS I-cache 失效时将如何处理？
13. 描述发生 MIPS D-cache 失效时将如何处理？
14. MIPS 4Ke 的 3 种操作模式有哪些？
15. 列出 ARM10E 处理器的主要特点。
16. 在 ARM10E 核中执行的有哪两个指令集？
17. ARM10E 流水线中指令流的分支意味着什么？
18. 列出 ARM10E 的 7 种操作模式。
19. 画图说明 ARM10E 的 6 站流水线。
20. ARM10E 中的外设和内存接口使用哪一种总线？
21. 列出 ARM10E 调试单元的 4 个主要特点。



## 参考文献

- ARM. 2002. *AMBA bus architecture specification*. ARM Limited. 参见本章第 13.2 节。
- ARM. 2002. *ARM1022E technical reference manual*. Revision: r0p2. ARM Limited.
- ARM. 2002. *ARM10 Thumb family, product overview*. ARM DVI 0014A, ARM Limited.
- Courtright, David. 2002. *MIPS32 M4K core for multi-CPU applications*. Embedded Processor Forum. MIPS Technologies.
- Francis, Hedley. 2001. *ARM DSP-enhanced extensions*. ARM White Paper. ARM Ltd.
- Kimelman, Paul. 2002. *Developing embedded software in multi-core SoCs*. ARM Ltd.
- MIPS. 2002. *Microprocessor architecture, performance headroom to compete*. MIPS Technologies.
- MIPS. 2002. *MIPS low-power synthesizable cores, a competitive comparison*. MIPS Technologies.
- MIPS32 4KE of synthesizable processor cores*. Product Brief, MIPS Technologies.
- MIPS32 4Kc processor core datasheet*. March 6, 2002. MIPS Technologies.
- MIPS32 4KE Processor core family software user's manual*. Document Number: MD00103, Revision 01.08, January 30, 2002. MIPS Technologies.
- Rowen, Chris. 2002. *Reducing SoC simulation and development time*. Computer, December 2002, reprint.
- Thekkath, Radhika. 2004. *Digital signal processing on the industry-standard MIPS architecture presentation*. Fall Processor Forum. MIPS Technologies.

## Tensilica 可配置 IP 核

- 本章目标：理解可配置处理器核技术

- 学习内容：

读者将会学习到 Tensilica Xtensa LX2 可配置处理器的体系结构和特性。

### 14.0 简介：再谈摩尔定律

1965 年，戈登·摩尔预言，集成电路的集成度将每两年翻一番。图 14-1 给出了标准单元密度与时钟频率随时间的变化图示，这种趋势的广泛认可和无情的抨击都使芯片开发人员不寒而栗。这种趋势使晶体管更加便宜、更加高速，但是也促使系统购买者期待功能、电池寿命、吞吐量和成本的不断优化。新功能的实现在技术上是可能的，问题在于如何实现。

CMOS 半导体尺寸不断减小，使得给定功能芯片的成本、面积、性能和功耗都同时得到改进。这种尺寸减小已经成为数字消费类电子、个人计算机和网络可以不断发展的首要推动力。

晶体管集成度的不断发展在 SoC 设计的发展中起到了关键的作用。不同的任务，例如音频和视频处理和网络协议栈管理，都可以相互独立的工作。具有固有并行性的复杂任务可以分解为紧密联系的一组子任务，这些子任务并行执行，与原始的非并行任务的功能相同。与采用单个计算资源串行执行相同的任务相比，这种类型的并发可以极大地改善应用延迟、数据带宽和功耗。

读者必须认识到，设计任务是有一定难度的。由于有 3 个原因共同作用，使得芯片的设计越来越困难。首先，半导体生产商对摩尔定律的跟踪取得了令人震惊的成功，设计者所要面对的门数每两年翻一番。其次，不断发展的工艺技术和电路特性，也推动芯片设计者在新的 IC 工艺技术开始执行之后，根据这些工艺进行设计。第三，可能也是最重要的，电子产品、消费者、计算和通信系统的市场需求也在不断发生变化，需要不断推出新的功能和性能，来刺激新的购买力。

结果，设计“高峰”越来越陡峭。当然，不断改进的芯片设计工具也帮助加快了 RTL 模拟速度，高容量的逻辑综合和更好的块布局布线技术也会降低一些设计难度；此外，针对系统逻辑设计重用的发展也可以降低新设计的工作量。然而这些改进都没有缩短设计难度的差距，这种公认的现象也体现在半导体研究公司对逻辑复杂性和设计者设计能力的简单比较上，如图 14-2 所示。

即使设计者努力跟上了先进芯片设计所需要的不断增长的产品需求，他们也必须面对另外的两种情况：

- 设计团队如何确保芯片说明书确实满足消费者的需求？

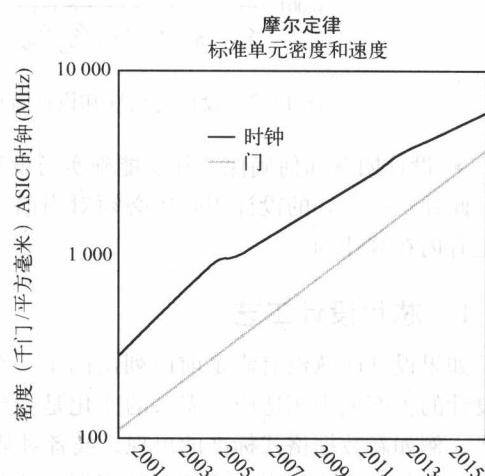


图 14-1 摩尔定律和门密度  
(经泰思立达公司许可使用)

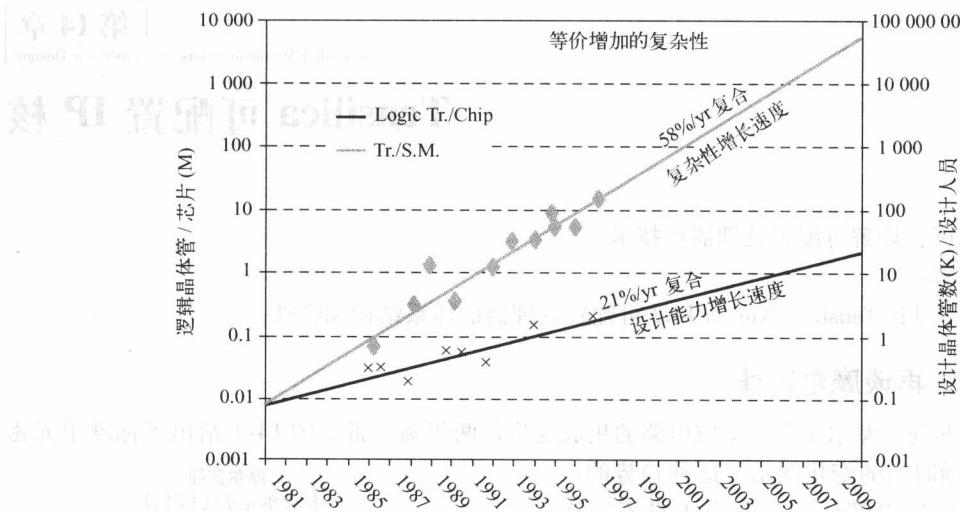


图 14-2 设计复杂性和设计者设计能力（经泰思立达公司许可使用）

- 设计团队如何确保芯片功能确实与设计说明书相符？

此外，一个好的设计团队还必须对当前用户的未来需求和未来的潜在用户进行分析，因为这是有内在规律的。

## 14.1 芯片设计工艺

如果设计团队没有满足前面列出的第一个标准，芯片可能会正确工作，但是销售量却不能与设计的生产成本相适应。需求的变化是受到特定的消费者驱动的，或者可以反映出整个市场趋势，例如新数据格式标准的出现，或者对某个产品类别期待出现的新特性。尽管大多数 SoC 设计都包含某种形式的嵌入式控制处理器，这些处理器有限的性能，通常使不会被用于基本的数据处理任务，因此软件通常不会用于增加或改变产品的新特性。

### 14.1.1 设计错误的芯片

如果设计团队没有满足前面列出的第二个标准，则必须进一步耗费时间和资源来修改和定位设计。这样会延迟芯片产品进入市场的时间，使得公司失去消费者。这种失败通常被认为是程序延迟。这种延迟的形式可以是由于综合或验证过程失败造成的，也可能是硬件 bug，没有在有限的验证覆盖率中发现明显的逻辑错误，这种验证通常是硬件模拟。

原因之一可能是存在于一个设计单元中的微小错误，或者是跟需求的沟通不足，软件和硬件团队、设计和验证团队，或者 SoC 设计者和 SoC 库单元或生产服务提供商之间在思想上的细微差别。无论是在哪种情况下，设计团队都必须进行重新设计、重新验证和芯片生产过程的循环。这种设计“反复”一般不会少于 6 个月，严重破坏了产品和商业计划。为了改进设计流程，必须能够同时在设计环境的 3 个相互作用的层次上进行同时改变，包括设计元素、设计工具和设计方法学。

设计单元是最基本的构造模块，硅结构、逻辑单元是形成设计的最基本要素。在历史上，这些模块是基本的逻辑函数（与非门和或非门，以及触发器）、用 C 语言和汇编语言设计的加法运算，通常应用在精简指令集计算机（RISC）微处理器和数字信号处理器上运行。

设计工具是应用程序和技术，设计者将其用于捕获、验证、提炼和转换特定任务和子系统的

设计描述。历史上，一些设计工具，例如寄存器传输级（RTL）编译和验证、代码汇编器和编译器、标准单元布局和布线，是复杂芯片设计的基本设计工具集。

设计方法学是设计团队的一种策略，用来将已有的元素和工具组合成系统的流程，以实现目标芯片和软件。设计方法学描述了都有哪些元素和工具可以使用，描述设计精炼的每一个步骤中要使用哪些工具，并列出设计步骤序列，如图 14-3 所示。当前主要的 SoC 设计方法学是围绕 4 个主要步骤来构建的，通常按下列顺序进行：软硬件划分；详细的 RTL 模块设计和验证；RTL 模块、处理器和内存模块的芯片集成；芯片制作后的软件设计。



图 14-3 EDA 和 IP 核合并（经泰思立达公司许可使用）

任何一个层次的改变都不会防止“构造错误的芯片”或“构造芯片错误”的 SoC 设计，在 RTL 设计或软件开发工具上的改进都不能解决更大的设计问题，而应该考虑的是有必要改变设计问题本身。设计元素、关键工具和无处不在的设计方法学必须同时改变。

### 14.1.2 SoC 设计的基本趋势

许多基本趋势表明，工程上需要一种 SoC 设计的新方法。第一个趋势是关于芯片集成度的急速增长，称为 21 世纪中设计电子产品的基本经济学。这个趋势的本质是这样的事实：在很长一段时间内，半导体生产商似乎希望并能够继续推进芯片的集成度，因为他们将持续地在许多方面进行创新——体积更小的晶体管、更小的互联尺寸、更高的晶体管速度、成本的大幅降低和低功耗。许多技术挑战，例如缩放比例问题、低功耗问题、纳米平版印刷、信号完整性及互联延迟等，都需要巨大的创新来解决。经验表明，在最差的情况下，这些条件最多只能放缓硅片按比例缩小的发展速度。

这种硅片按比例缩小的趋势又引发了第二个趋势，即按照现有的集成度，将一个电路集成到一个硅片上。这个趋势是由于对现代电子产品要求多样性和多功能性而产生的。不断增长的集成度，使得能够将所有关键功能与网络交换机、数码相机或个人信息工具相结合，并将这些功能放入到一个硅片中。也就是说，将所有的逻辑、存储器、接口，实际上几乎所有终端产品中能够包含的所有电子功能，都放入到一个硅片中。

芯片集成度的提高带来的优势是非常明显的。更紧密的集成可以改变终端产品的构成形式，使得复杂的系统足够小，可以放入口袋中、电视中或者汽车上。更高的集成度还可以进一步降低功耗，使得更多终端产品能够用电池供电，可以不用风扇；可以使电池供电的产品能够适应各种不同的工作环境。不断增长的集成度也促进了产品性能的提高，这种性能可以是根据产品完成处理的速度来衡量的，或者根据一个产品中可以集成的功能数量来衡量。实际上，这些属性会成为非常重要的产品属性，可以使普通消费者涌向零售商，购买新的产品来替换他们手里的旧产品。

### 14.1.3 每个系统都采用一个新的SoC实现是不现实的

芯片集成度的不断提高也对产品设计者提出了经济挑战。如果一个系统中的所有电子功能都包含在一个芯片中，则该芯片会成为设计者希望定义的终端产品的直接反映。但是，这样的芯片设计缺乏灵活性，不能广泛应用于各种产品。

由于高度集成的芯片缺少一些特性，来促使其更加灵活、可复用性更强，因此SoC设计的趋势是，在芯片设计和系统设计之间存在直接的1:1对应关系。最终，如果SoC设计沿着这条路走下去，开发一个新系统的时间，以及构建一个新系统所需要的工程资源数量，将至少与设计一款新的芯片所耗费的时间和成本相同，甚至更多。有一种创建灵活SoC的方法，使其可以适用于多种应用，如图14-4所示。

如果一个灵活的SoC设计可以用于多种应用，就可以极大地节省成本，图14-5对这种情况进行了详细说明。从图中可以很容易看出，由于应用量很大，前期前段的设计成本，会完全由单芯片成本的价格降低来补偿。

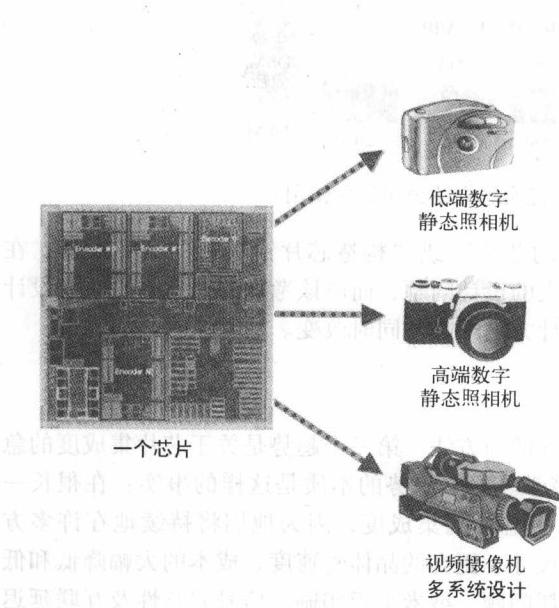


图14-4 单SoC多应用  
(经泰思立达公司许可使用)

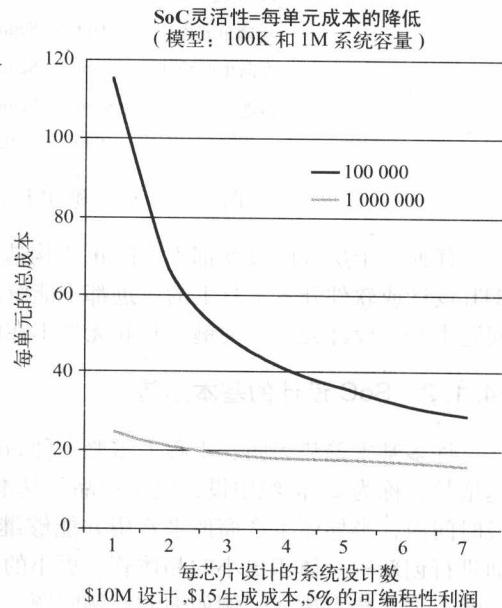


图14-5 SoC灵活性与单芯片成本的对比  
(经泰思立达公司许可使用)

在过去，产品设计者将许多芯片组合在一个大的印制电路板(PCB)上，来构建系统。不同的系统使用不同的芯片组合，将它们焊接在系统专用的PCB上，由于可以使用各种芯片部件，并且因为PCB设计和原型构建比较简单，这种方法是可行的。相对来说，系统的可编程性并不重要，因为相对来说对系统进行重新设计是比较便宜的，速度也比较快。

### 14.1.4 纳米技术

在纳米半导体技术中，情况完全不同。人们追求更小的系统尺寸、更高的能耗效率、更低的生产成本，这些都使得尺寸巨大的PCB不再符合要求。并且，面向数量的终端产品需求只能通过SoC设计来得以满足。即使是适当的“虚拟部件”，也可以作为SoC的构建模块，SoC设计集成和原型构建的费用也要比PCB设计和原型构造多两倍以上。此外，SoC设计修改通常要用几个月的时间，而PCB的修改只需要几天的时间。SoC设计是发挥纳米技术优势的主要手段。但是为

了使 SoC 设计更加实际, SoC 的构建方法不能等同于 PCB 的构建方法。

SoC 不灵活的缺点必须解决, 因为芯片级的不灵活对于芯片硬件设计的可重用性确实是一个危机。尽管工业界对模块级硬件复用 (IP 复用) 给予了很大的关注, 但模块内部复杂性的发展和模块间的复杂连接, 已经限制了 IP 模块系统的和经济的复用。图 14-6 给出了 SoC 硬件和软件的设计过程。

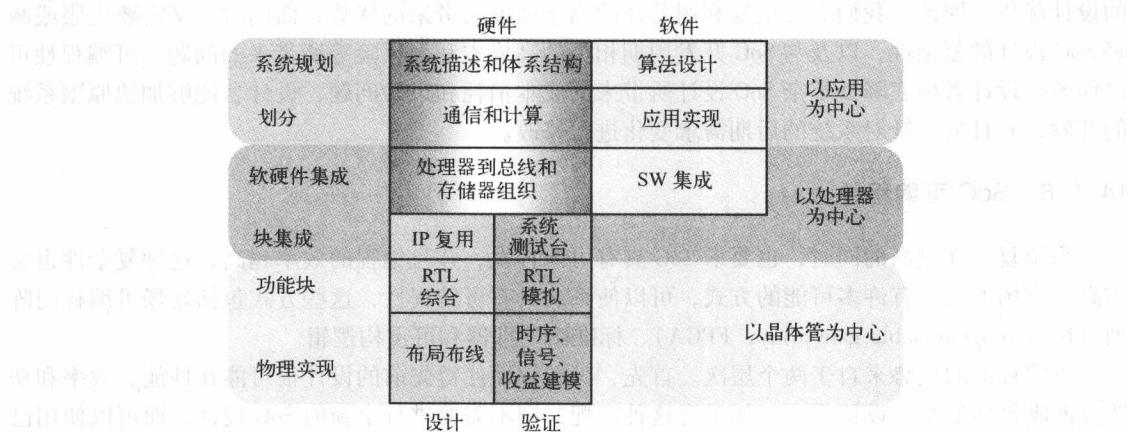


图 14-6 SoC 设计流程 (经泰思立达公司许可使用)

通常, 用户需求、实现的标准和其他功能必要的接口, 必须随着产品的改变而得到发展。这些将成功的模块复用可以归为两种类型——实现了稳定接口功能的简单模块、可以在处理器中实现的内建灵活功能, 这些功能的灵活性和可适应性是通过软件的可编程来实现的。

另一方面, 如果需要为每一个系统都设计一款新的芯片, 这对于系统开发人员来说在经济上是不可想象的, 因为芯片的设计是非常困难的。我们可以通过使用好的芯片设计工具来改善这种状态, 但是如果缺少芯片设计方法学方面的巨大创新, 这种状态的改善也不会很快。

实际上, 如果缺少一些主要的创新, 设计一款芯片所要付出代价的增长速度, 比芯片本身晶体管复杂性的增长速度要快得多。因为缺少在设计方法学方面的创新, 我们在系统设计方面将会面临失利的状态。一旦处于这种状态, 情况将是不可想象的, 因为系统设计和芯片设计正在变得越来越接近。

### 14.1.5 SoC 设计改革

系统开发人员必须解决如下两个问题:

- 通过降低系统中所用芯片的设计难度, 实现用越来越少的资源开发系统的目的。
- 增强 SoC 的可适应性, 这样就不必为每一个新的系统设计一款新的 SoC。

解决上述两个问题的一种方式是, 使 SoC 充分地可编程, 这样一款芯片设计将可以用于 10 或 100 甚至 1000 个不同的系统设计, 而只付出很少甚至不付出集成方面的工作。解决这些问题意味着可以用已经生产的芯片来满足下一个系统设计的需求, 用大量的系统设计来补偿新芯片开发所付出的努力, 如图 14-7 所示。

这种趋势形成了需要对 IC 设计进行基本改变的推动力。这种基本改变, 不仅能够极大地降低 SoC 的设计难度 (不仅在

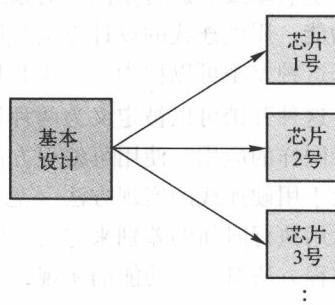


图 14-7 多系统设计

于芯片设计，还在于所需要的软件），而且增加 SoC 设计固有的灵活性，这样可以在多个系统设计之间共享芯片设计所付出的努力。

电子工业在经济方面所取得的成功，关键在于能够使未来的 SoC 更加灵活，同时性能也获得极大的提高。SoC 工业和 SoC 设备用户的核心困难在于如何同时对灵活性和最优性之间进行很好的折中。SoC 设计者希望降低芯片设计成本，而同时又能够利用不断提高的芯片集成度获得更多的设计优势。因此，我们需要重复利用芯片的高集成度所带来的优势，而同时，又需要克服或减轻 SoC 设计的复杂性，以及与 SoC 开发周期相关的高成本和高风险等所带来的问题。可编程性可以使 SoC 设计者彻底降低复杂 SoC 设计所带来的成本消耗和风险问题，设计者能够加快原型系统的开发，并且易于针对系统的后期需求变化进行修改。

#### 14.1.6 SoC 可编程性

系统复杂性提高的同时，也要求 SoC 具有可编程性，并且编程的效率越高，这种复杂性也会越高。市场上已经有许多可能的方式，可以使系统具有可编程性，这些方式包括现场可编程门阵列（field-programmable gate array，FPGA）、标准微处理器和可重构逻辑。

可编程性的优势来自于两个层次。首先，可编程性使得先前的设计很可能在性能、效率和功能方面满足新的系统设计需求。如果有这种匹配，则不需要进行全新的 SoC 设计，而可以使用已有的平台。其次，可编程性意味着即使必须进行全新的 SoC 设计，其中的许多功能也可以可编程的方式实现，降低了设计风险和难度。FPGA 和处理器市场所取得的成功都是可以利用的。

不同平台的编程模型差别很大。尽管复杂性的提高会极大地影响性能，但传统的处理器（包括 DSP）所执行的应用复杂性是没限制的。处理器通常使用先进的流水线技术和电路设计技术，以获得高的时钟频率，但是所获得的并行性却一般，每一个时钟周期只能执行一个（或多个）操作。而与之相反，FPGA 的容量是有限的。一旦问题的复杂性超过了一定的程度，将完全不适合 FPGA 处理（见图 14-8）。

而另一方面，FPGA 可以以很高的并行性来实现算法，有时在一个时钟周期内可以执行数百个操作。另外，FPGA 的工作频率通常不高，并且对于相同的应用，比处理器的芯片面积和芯片成本都要高。

#### 14.1.7 可编程性与效率对比

可编程所带来的好处，都使隐含的芯片设计具有一般性，而同时允许用户根据特定的情形，在系统启动或者运行期间，对系统进行配置或个性化设置。传统可编程性的问题在于，对于相同的功能，用硬连线的设计方法与用可编程技术的设计方法相比，在效率和/或性能之间的差距太大。这种差距可以称为“可编程性开销”。

这种开销可以被定义为两种设计方法之间的面积差别，即对于同样的应用，使用可编程方法对一个功能的实现，其面积要大于用硬连线的实现方法（见图 14-9）。此外，这种开销也可以用执行时间的差别来定义，即对于同样的硅片面积，使用可编程方法对一个功能的实现，执行时间要大于用硬连线的实现方法。

FPGA 或通用处理器的可编程性所带来的优势，是 10 倍以

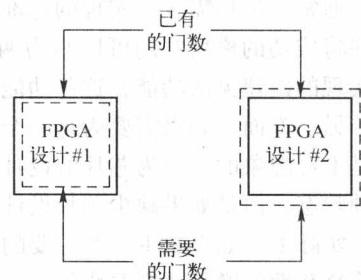


图 14-8 FPGA 门数的限制

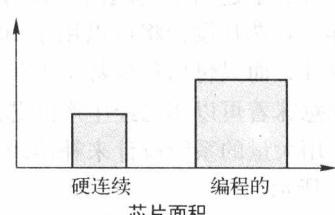


图 14-9 可编程性差距

上，甚至可以达到百倍。例如对于安全应用，如DES和AES加密，在任务相同的情况下，采样硬连线的设计方法与采样通用RISC处理器执行的方式相比，速度通常要快几百倍。这种加密功能的FPGA实现，其时钟频率通常比硬连线的逻辑要慢3~4倍，但是其芯片面积则要大10~20倍。

这种效率的差别是由数字基础的一般性决定的：FPGA和通用处理器。通用处理器的设计者在设计时考虑的是，所构建的平台是否能够满足所有可能的情形。然而不幸的是，真正通用处理器的设计，需要丰富的基本信息进行支持，例如用哪些器件实现特定的计算功能，以及在计算过程中移动数据的连接通路如何实现等。芯片的效率受到了有限复用的限制，或者说是受到实现应用基本功能的晶体管“分时复用”的限制。

实际上，如果查看FPGA或者通用处理器执行“加法”计算的过程，就可以发现，这组逻辑门是由一个加法器及其周围的大量多路器和线路构成的。这些线路和多路器用于在正确的时刻，将正确的数据送到正确的加法器上。与保存和移动数据、选择正确的功能序列相关的电路开销，比起那些操作序列已知的设计来说，其电路延迟、晶体管数量和线路数量都要大得多。

通用处理器要依赖基本功能单元的分时复用，来完成基本算术和逻辑运算以及内存访问操作。大多数处理器逻辑都用作连接硬件，用于将不同的操作数连接到少数共享硬件功能单元上。在不同的操作之间重用处理器寄存器和内存单元，这样可以隐含的实现功能单元之间的通信；与之相反，FPGA逻辑一般不在不同的功能之间共享硬件。每一个功能都是静态地映射到FPGA芯片的特定区域上，因此每一个晶体管通常只反复执行一项功能，如图14-10所示；功能之间的通信通过功能之间互联的静态配置来实现。

关于需要的计算了解得越多，就可以有越多的计算过程中所需要的晶体管与专门的连线对进行互联，以获得较高的计算单元利用率。通用处理器和通用FPGA技术都有开销，对软件可编程性的探索，突出了先前隐藏的现场硬件可编程性的开销。

现代软件可编程性的能力，来自于两种互补的特性，其中一种是抽象。软件程序可以使开发者以更加简洁、更加易懂的形式来处理计算问题，并且可以与实现细节无关的情况下进行算法改进。抽象技术将实现细节隐藏，可以更加全面地考虑实现方法的结构。

人数中等的软件设计团队通常要设计数百上千行的源代码来开发、重用和改进应用程序，包括广泛的操作系统重用、应用程序库的重用和中间件软件组件的应用等。此外，人们还开发了复杂的应用程序分析工具，用于帮助设计团队调试和维护这些复杂的应用程序。相比之下，人数相当的硬件设计团队使用数万行的Verilog或VHDL代码来设计逻辑功能，设计目标更大、更复杂。只有倍加小心才能对模块进行修改。编码的抽象仅限于简单的寄存器、存储器和简单的算术功能，这些可能构成模块中的基本可重用硬件功能。

第二个特性是软件的修改比较容易。当第一次启动系统时，系统的功能就发生了变化；并且随着任务的切换，系统功能也动态地发生变化。在软件驱动的环境中，如果需要将一项任务彻底修改为一个子系统的功能作为响应，系统可以在数微妙之内从内存中为该子系统装载一个新的基于软件的功能块，来改变系统的需求。这是非常关键的一点：软件灵活性的经济优势，不仅体现在开发周期过程中（从概念到系统“上电”），也体现在设计的预期生命周期中（产品说明确定最后的产品修改并提供给最后的用户），如图14-11所示。

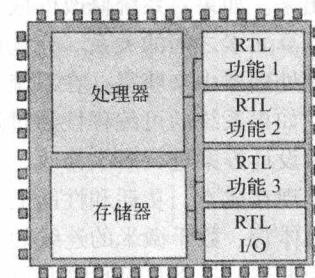


图14-10 固定的FPGA功能  
(经泰思立达公司许可使用)

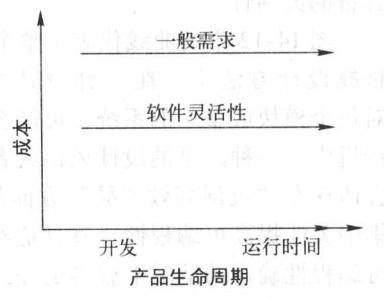


图14-11 产品生命周期成本

对产品的需求要保持适应性，这对提高产品的收益性是非常关键的。如果可以快速并以较小的代价对系统进行修改，设计者就会降低因为不能满足设计说明的需求而带来的风险，因此就会有更多的机会快速根据新用户的需求对产品进行修改。现场升级软件，对于PC系统来说是比较常见的，能够现场对软件进行升级的特性也开始应用到嵌入式产品中。例如，Cisco 网络交换机就可以进行现场软件升级。灵活性越大（成本给定的情况下），用户数量就会越多，SoC 的产量就越大。

与之相反，硬连线的设计选择必须在系统设计周期开始时就进行。如果在产品开发周期中的某个时间，例如设计、原型构建、现场检验、现场升级或者在进行第二代产品的开发时，系统设计者决定要修改关键的计算或通信策略，则会等价于重新开始系统的设计。硬连线设计决策的重要性也限制了 SoC 产品可以适用的用户和系统范围，进而限制了产品的产量。

增强系统的可编程性有优势，但是也要付出代价。这种优势可以在开发的灵活性和效率中看出来。如果对系统修改的成本比较低，设计者在设计周期的起始阶段，没有必要准确地决定各个计算元素之间的关系。究竟可编程性是通过将网表下载到 FPGA 中来获得，还是通过将软件下载到处理器中来获得，在系统上电之前，设计者不用对系统的最终配置做出决策。

如果系统的可编程性通过运行在处理器上的软件来体现，在产品设计好之前，开发人员可以对设计方案进行多次修改，一些设计决策的改变甚至会发生在产品生产前夕。可编程性的代价体现在成本、功耗和性能上，传统的处理器和 FPGA 可编程性，带来了计算功能模块之间数千个晶体管、数千微米的连线的开销。对于相同的功能，与硬连线实现方式相比，这种开销最终转化为 FPGA 芯片面积的增大和工作频率的下降，以及处理器执行时间的增加。

将实现硬件固定制作在硅片上，可以极大地改善单元成本和系统性能，但是同时也极大地增加了设计风险和设计成本。设计团队面临着决策：哪些功能需要用硬件实现？哪些功能用软件实现？哪些功能将来更容易发生变化？模块间的通信关系将如何发展？

这种效率和性能以及可编程性和灵活性的折中，是本书中会反复重复的主题。

图 14-12 给出了这种折中的概念。

图 14-12 中的纵轴表示模块或整个系统的内在复杂性，横轴表示模块或整个系统的性能或效率需求。需要反复面对的 SoC 设计选择是：具有足够的灵活性的解决方案，可以支持复杂的设计，但会牺牲性能、效率和吞吐量；具有更高效率或吞吐量的解决方案，通常会牺牲处理复杂应用必须具备的灵活性。

图 14-12 中的曲线代表了整个的设计风格或设计方法学。在一种设计方法学中，对每个模块或整个的系统，可能会有多种实现方法，但是设计团队必须在复杂性和效率方面进行折中。一种改进的设计风格或者方法学，可能仍然会体现在解决方案上的折中选择，但是最终会体现在“灵活高效产品”方面的改进。在向改进的设计方法学转变的过程中，设计团队可能集中关注提高可编程性（在性能给定的情况下获得更好的灵活性），或者集中关注提高性能（在可编程性确定的情况下获得更好的效率和吞吐量）。在某种意义上，高效 SoC 设计的关键在于对不确定性的处理。如果对于那些要求稳定的产品，设计团队可以对产品的所有方面进行优化；而

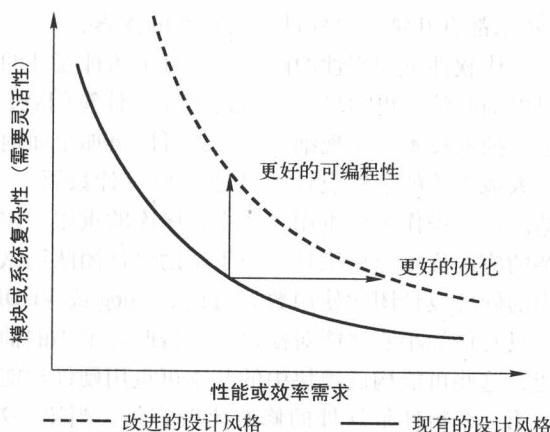


图 14-12 基本的设计折中（经泰思立达公司许可使用）

对于那些要求不稳定的产品，让产品的所有方面具有更强的灵活性，将会比他们的竞争者获得更加便宜、更加持久、更加高效的产品。

#### 14.1.8 SoC 设计成功的关键

SoC 开发的目标是在两个方面获得一种最优的平衡，其中一方面是获得 SoC 足够的灵活性，来满足某一个问题的需求变化；另一方面是针对一个最终应用，仍然能够实现高效率和最优化。因此，就需要一种 SoC 设计方法学，能够实现系统和子系统的高级并行性、适当的可编程性和设计过程的快速性。基于 SoC 系统的开发者没有必要开发完全通用的芯片，产量足够大的 SoC 芯片可以是专用的。例如，数码相机的设计者不需要与高端光学网络交换机使用一样的芯片。然而，一款相机芯片应该对消费类图像处理产品提供广泛的支持，如图 14-13 所示。

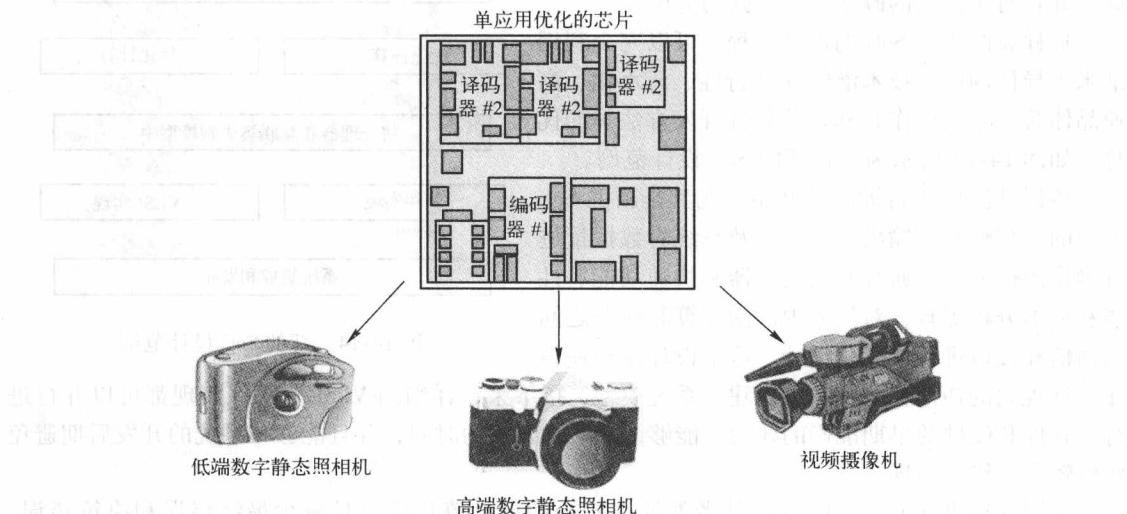


图 14-13 可以用于多种相机系统中的一款相机 SoC

10 种类似的设计共享一款芯片，和 1000 种设计共享一款芯片，这两种情况下所获得的收益相差不大。如果每一种相机的产量是 200 000 个，共享的 SoC 设计成本是 \$10M，则 SoC 设计给最终的相机成本的贡献为 \$5 (-5%)。在 1000 中产品中共享 SoC 设计将节省 \$5 的设计成本；但是在设计 SoC 时需要考虑一般性，而这种设计成本的增加远远超过 \$5。SoC 设计并不需要完全通用，因为高产量的产品构建一个适用于应用领域的芯片级设计平台是值得的，而在平台上具有很好的设计灵活性。

如果设计人员在设计 SoC 时，要考虑将来在产品的生命周期中可能遇到的所有应用，这样就需要 SoC 具有足够的灵活性，那么他们会获得与全局灵活性相关的所有优势，而不需要付出使芯片具有完全一般性的开销。如果平台设计是正确的，这种专用灵活性的成本，要远远低于从通用设备所获得的灵活性，例如 FPGA 和高性能、通用处理器相比。

此外，一种好的设计方法学能够广泛地在硬件和软件工程师中流行，让他们都能够对 SoC 进行设计和编程。掌握技术的人力资源越丰富，产品的开发速度会越快，成本也会越低。

这种 SoC 设计方法学的主要特点如下。

- 支持并发处理。
- 适当的应用效率。

即使对于不是 SoC 设计专家的人来说，也易于进行相应的开发。但这并不是说采用该设计方

法学进行 SoC 开发的人可以不是 IC 设计专家，而只是说他们可以不是特定应用领域方面的专家。

### 14.1.9 改进的设计方法学用于 SoC 设计

加速百万门级 SoC 开发的一种主要方法正在发展之中。首先，处理器替代了硬连线逻辑，加速硬件设计，并使系统具备芯片级的可编程性。其次，应用到 SoC 中的处理器通常被自动扩展，能够高效的运行，而同时具有高吞吐量、低功耗和较低硅片面积的特点。基于扩展处理器的模块通常具有一些特性，其性能能够与被替换的 RTL 模块相当。并且，这些处理器成为 SoC 的基本构成模块及处理器的开发迅速、接口灵活、编程和调试简单等特性都加速了整个设计的过程。最后，可能也是最重要的，最终基于 SoC 的产品效率很高，并且对于需求的改变具有很强的适应性。

这样就改进了 SoC 的设计流程，可以完全利用纳米半导体的内在技术潜能（并行性、流水化、高速晶体管、专用操作）和现代软件开发方法学的优势。如图 14-14 所示为一种新的 SoC 设计范例。

该设计流程从高级需求开始，尤其是新的 SoC 平台的外部输入和输出需求，以及系统对数据量进行操作的任务组，通过专用处理器和快速功能、性能和成本分析工具，对任务内部的计算和任务之间的通信和接口进行最优化设计。这个设计流程使得

在设计规划的初期就能够准确地建立系统模型，接下来，详细的 VLSI 和软件实现都可以并行进行。软件和硬件的早期准确的建模，能够缩短系统开发的时间，并且能够在系统的开发后期避免代价较大的错误出现。

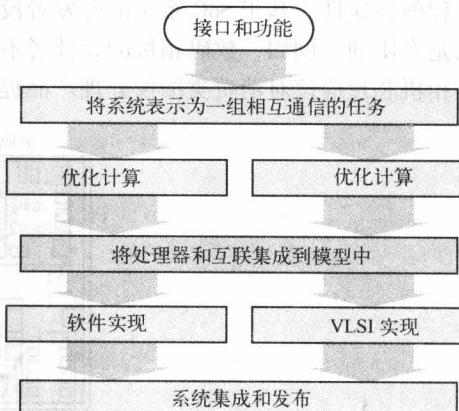
使用这种设计方法意味着设计者能够在整个的设计流程中尽量减少最终错误和决策错误，并且不需要设计反复。这意味着 SoC 设计者能够在设计周期的早期进行比较全面的、详细的设计可能性分析。采用这个方法，设计者可以更好地理解设计的硬件成本、应用程序的性能、接口、编程模型以及 SoC 设计中的其他重要特性。

采用这种方法设计 SoC，意味着使用这种半导体平台的可能性越大，在成本和功耗效率之间进行最小折中的可能性就越大。设计团队越多使用专用处理器作为 SoC 的构建模块，而不是使用 RTL 编写的硬连线逻辑，SoC 将越有可能发掘以软件为中心的设计方法的内在灵活性。

### 14.1.10 可配置处理器作为构建模块

在这种方法中，基本的构建模块是一种新型的微处理器——可配置、可扩展的微处理器核。这些处理器是由发生器创建的，这种发生器将高级应用领域的需求（以指令集描述的形式设置是以应用代码例子的形式）转换为高效的硬件设计和软件工具。SoC 设计的“处理器海”方法使得没有微处理器设计经验的工程师，也能够对这些基本的构建模块进行描述、评价、配置、编程、互联和组合，使它们成为处理器的组成部分，共同产生 SoC 设备的基本数字功能。

为了使用这种可配置微处理器核进行处理器配置的开发，芯片设计者及应用程序专家首先要面对处理器 - 发生器接口（见图 14-15），并选择或描述应用资源、指令集选项、存储器等级、紧耦合的外设和应用所需要的接口。完全产生标准 RTL 语言、EDA 工具描述和测试台形式的硬件设计，以及软件开发环境（C 和 C ++ 编译器、调试器、模拟器、RTOS 代码和其他支持软件），需要大约一个小时的时间。



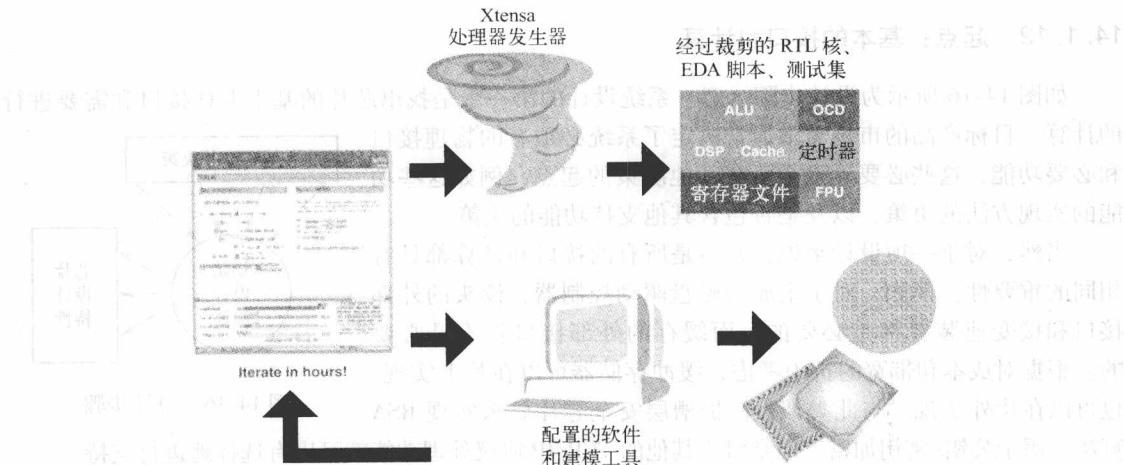


图 14-15 基本的处理器设计流程

产生过程结束后能够马上获得厂家可以接受的硬件实现和软件开发环境。这种硬件和软件基本结构的及时产生，可以针对处理器设计，对软件应用程序进行快速地修改和测试。软件的完整性极大地消除了软件移植的问题，可以加速设计的反复过程；还可以对工具进行配置，使其以某个普通的应用为基础，自动产生许多可能的处理器配置。根据这种应用需要的目标来衡量，能够表现出更好的硬件解决方案。

专用处理器能够执行微控制器或者高端 RISC 处理能够执行的所有任务：运行用高级语言编写的应用程序，实现各种实时性，支持复杂的协议栈、库和应用层。即使是按照传统的微处理器功耗、速度、面积和代码数量的标准来衡量，专用处理器仍然能够非常高效地执行一般的整数处理任务。但是由于这些专用处理器能够根据嵌入式应用所提出来的特质数据类型和计算要求，提供数据通路、指令和寄存器存储，实质上它们还可以支持芯片设计者传统用硬连线逻辑实现的所有功能。

#### 14.1.11 使用自动生产的处理器快速进行 SoC 开发

如果要创造最佳的 SoC 结构，系统体系结构面临着一系列重要的决策。在设计阶段的早期所做的正确选择可以降低芯片成本和功耗，提高系统性能，并提高开发和验证的效率。这种设计流程鼓励系统设计者广泛使用处理器作为默认的执行任务方法，并在 SoC 设计的框架内集中考虑如何在成本、性能和灵活性之间寻求平衡。这种设计流程的基础有如下几项：

- 从系统的基本 I/O 接口和计算需求开始，自顶向下的进行设计。
- 当任务具有特定的计算模式时，广泛地使用处理器对任务进行处理，并对处理器进行优化，以满足任务的要求。
- 如果一项任务超过了已优化的处理器的处理能力，可以在多个处理器之间对任务进行并行处理。
- 如果一个处理器的能力可以同时处理多项任务，可以将这些任务都映射到一个处理器上，将硬件的成本、功耗和通信开销降到最小。
- 估计通信模式，并围绕这些模式，对软件和硬件进行优化。
- 尽早对通信任务进行粗略的模拟，并将系统的实现细化到处理器、软件和其他的模块，并且模拟会随着设计的进展而逐渐准确。

### 14.1.12 起点：基本的接口和计算

如图 14-16 所示为设计步骤示意。系统设计的第一步是找出芯片的基本 I/O 接口和需要进行的计算。目标产品的市场需求通常决定了系统必须有的物理接口和必要功能，这些必要元素是所有其他决策的起点，例如这些功能的实现方法的决策，以及是否包含其他支持功能的决策。

当然，对于一项设计来说，并不是所有的接口和计算都具有相同的重要性。例如，对于集成的磁盘驱动控制器，读头的外部接口和接变速装置都是必要的，而缓存的外部接口就不是必要的。根据对成本和带宽的折中考虑，缓冲存储器可以在片上实现，也可以在片外实现。与此类似，保护槽层安全芯片要求实现 RSA 算法，用于公钥/密钥加密，但是对于其他的 TCP/IP 协议处理功能就可以有选择地进行支持。

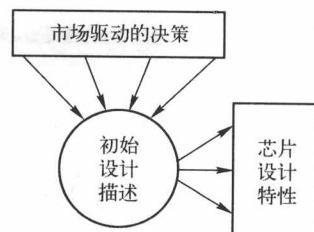


图 14-16 设计步骤

### 14.1.13 并行处理任务

如果某个任务有特殊的处理需求，就要求设计者必须选用速度较快的通用处理器，或者采用并行度比较高的硬件实现方式。高速的通用处理器通常需要很高的时钟频率，例如，大于 3GHz。时钟频率较高的处理器通常既不利于降低功耗，也不利于设计和集成到 SoC 中，除非使用特殊的设计方法。对于嵌入式的应用，采用并行硬件资源的方法更适合于嵌入式的 SoC 需求。

在过去，采用基于 RTL 的硬连线逻辑设计，是实现并行设计的唯一选择，但是这种设计方法能够实现的算法复杂性是有限的。可扩展的处理器给出了一种开发并行性既简单又高效的方式，尤其是对于细粒度（指令集）并行性来说。

基本的算法分析和指令设计流程是系统的，但也是复杂的。用来设计专用指令集的类似于编译器的工具可以提高这个过程的自动化程度。实际上，许多先进的代码选择编译器算法、软件流水线寄存器分配、长指令字操作的处理调度方法，也可以用于发现和实现新的指令定义，并用这些指令产生代码。自动处理器生成的基本流程建立在专用处理器生成的基础上，但是增加了自动处理器体系结构生成，如图 14-17 所示。

然而，自动处理器发生所带来的好处远不仅限于发现改进的体系结构。与人工设计的指令扩展方法相比，自动工具不需要向应用程序代码中人工加入任何新的数据类型和固有的功能，能够有效地加速那些对于人类程序员来说太大、太复杂的应用程序的开发。

因此，在将处理器体系结构的开发转换为嵌入式应用的开发时，这种技术是非常有希望的。自动处理器生成的基本目标有如下几项：

- 对于能力一般的软件开发人员，可以很容易地掌握工具，并一直获得很好的结果。
- 在不需要进行源代码修改的情况下，可以利用产生的指令集。在暴露隐含的并行性方面，一些算法的表达方式比较好，尤其是对于 SIMD 优化，这时就可以使用代码调整。自动处理器发生器可以对于开发人员对代码的改动给出一些指导。
- 产生的指令集必须具有足够的通用性和鲁棒性，这样在对应应用程序代码进行少量修改之后不会降低应用程序的性能。体系结构设计自动化环境应该提供适当的指导，使得高级开发人员能够进一步提高自动产生的指令集扩展，从而获得更高的性能。
- 开发工具必须具有相当的速度，可以访问大量的潜在指令集扩展，按照每分钟数千条指令为标准。

由于对系统一般性和可重编程性的要求，需要系统具有如下两个相关的使用模型：

- 初始的 SoC 开发：输入 C/C++，输出指令集描述。

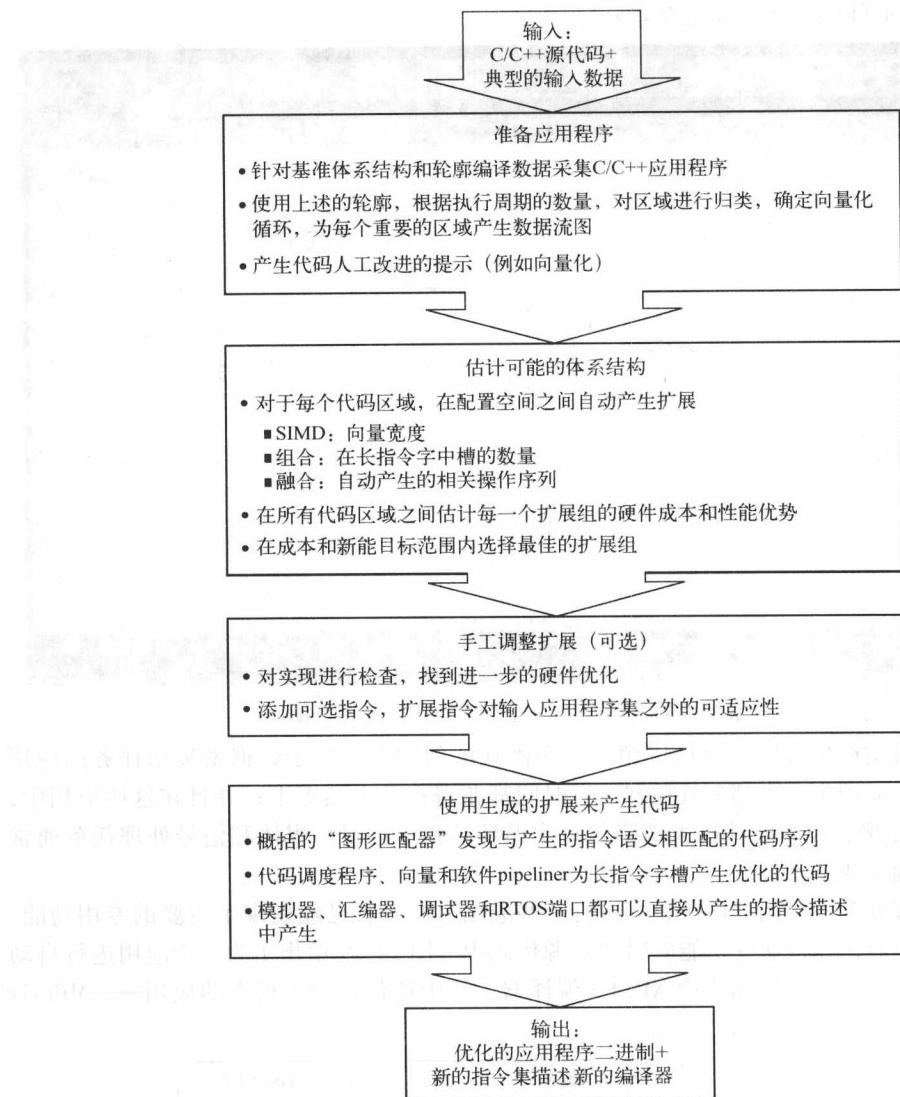


图 14-17 Tensilica XPRES 处理器生成流程（经泰思立达公司许可使用）

- 对现存的 SoC 进行软件开发：输入 C/C++ 和产生的指令集描述，输出二进制代码；Tensilica XPRES (Xtensa 处理器扩展系统) 编译器实现自动的处理器指令集生成。

更加详细的 XPRES 流的解释能够帮助解释这种更进一步的处理器自动化的使用和功能。图 14-17 给出了 XPRES 编译器实现的 4 个步骤，除了可选的人工步骤之外，其他所有步骤都是机器自动化的。

裁剪的 C/C++ 编译器的产生，进一步增加了自动产生处理器的用处。即使是在源应用程序发展的时候，合成编译器也会极力寻找使用扩展指令集的机会。实际上，这种方法对于产生通用体系结构也是非常有用的。只要基本的操作集合适用于另外一个应用，即使是前后两个应用程序没有分行关系，合成编译器通常也能有效地利用扩展的体系结构。

自动处理器发生器在内部列举数千种配置中每一种的估计硬件成本和应用程序性能优势，然后创建 pareto 曲线，如图 14-18 所示。曲线上每一个点代表了每一级增加的门数所获得的最高性能等级。该图来自 Tensilica 的 Xplorer 开发环境的屏幕截取图，该环境正在对简单的视频运

动估计子程序运算的 XPRES 结果（绝对差分）。

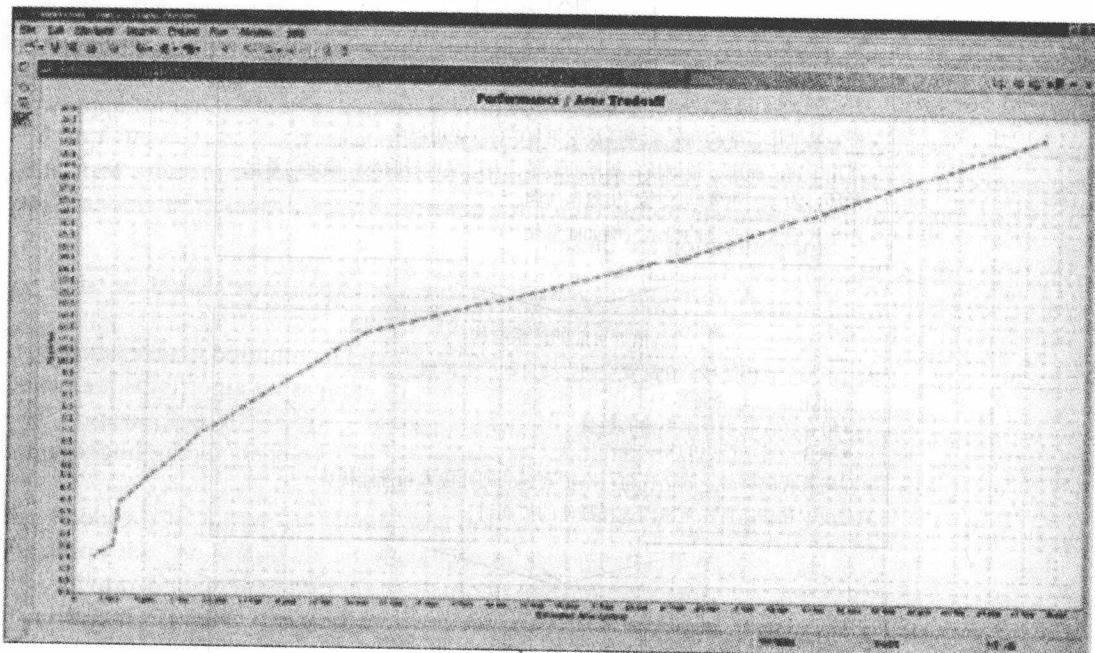


图 14-18 体系结构的自动生成（经泰思立达公司许可使用）

指令集扩展的自动产生可以广泛地应用到许多潜在的问题上。对于数据密集型任务的应用效果最好，在这样的应用中，大部分处理器执行时间都花费在几个热点上；并且在这些应用中，SIMD、宽指令和操作聚合技术能够极大地降低每次循环的指令数量。媒体和信号处理任务通常直接由自动体系结构生成来实现。

自动生成器能够处理的应用程序中，其中有一些应用，开发者已经发现了主要的专用功能，用 TIE 实现了这些功能，并将这些功能应用到 C 源代码中。图 14-19 给出了为三个应用进行自动处理器生成的结果，这三个应用都使用 XPRES 编译器，其中还包括一个很大的应用——MPEG4 视频编码器。

应用	MPEG-4 编码器	Radix-4 FFT	GSM 编码器
加速	3.0x	10.6x	3.9x
基准代码大小	111KB	1.5KB	17KB
加速后的代码大小	136KB	3.6KB	20KB
评估的配置	1 830 796	175 796	576 722
发生器运行时间 (分)	30	3	3

图 14-19 应用比较

图中还对每一个应用给出了两种情形下的代码大小结果，分别是基准 Xtensa 处理器体系结构和自动优化的 Xtensa 处理器体系结构。使用优化过的指令集通常会增加代码的数量，但是在任何情况下，与传统的 32 位 RISC 体系结构相比，优化后的代码仍然要小得多。图中还给出了进行评估的配置数量，随着应用程序大小的增加，配置数量也增加。自动处理器发生器的运行时间也随着应用程序大小的增加而增加，但是在主频为 4GHz、运行 Linux 的 PC 上，一般会达到每分钟 100 000 种评估配置。

图中还给出了一个合成体系结构的一般性示例。在编译和运行 GSM 编码器源代码时，并没有使用针对 GSM 编码器进行优化的体系结构，而是使用针对 FFT 优化的体系结构。尽管两种都

是 DSP 分割的应用，但是在源代码上没有共同点。不过，针对 FFT 优化的体系结构自动合成的编译器，仍然能够找到处理器的 FFT 优化指令集。与针对基线 Xtensa 处理器指令集编译的代码相比，GSM 编码器的运行速度提高了大约 80%。

全自动的指令集扩展有如下两个重要的注意事项：

- 程序员可能知道他们所设计的应用程序的一些行为，而这些行为并没有在 C 或 C++ 代码中明确地表达出来。例如，程序员会知道一个遍历的取值范围，或者两个非直接引用的数据结构永远都不会重合。
- 如果不知道源代码的这些信息，可能会抑制机器代码和指令扩展中的自动优化。使用自动指令集发生器的指南中应该给出一些的线索，以便更好地将这种应用程序特有的信息结合到源代码中。进行指令可扩展的人可能会知道这些信息，并且发掘这些附加的信息来创造指令集，并进行相关的代码修改。

对于一项任务，专家和程序员有时会开发出完全不同的新算法。比起最初在 C 和 C++ 源代码中的算法，采用不同的内循环算法可能更有助于加速指令的执行。很可能总是存在这样一类问题，尽管人类开发更优化的体系结构会需要较长的时间（有时候会长得多），但专家的处理效果会胜过自动发生器的处理效果。

#### 14.1.14 自动指令集发生的含义

自动指令集发生的实现具有广泛的含义。首先，这种技术使许多设计者都可以创建专用处理器，并且甚至不需要对指令集体系结构有基本的理解，只需要能够运行编译器，就可以利用自动指令集扩展机制。

其次，自动指令集发生可以有效地处理复杂问题。关于这些问题，应用程序性能的瓶颈存在于多个循环或代码段之间。一种自动的、基于编译器的方法，能够很容易发现各个循环之间共享指令的隐含情况，基于动态执行来发现不同代码段的相对重要性，以及硬件成本估计。对于人类设计者来说，全局优化的难度会更大。

第三，自动指令集发生能够确保在不需修改源代码的情况下，可以将新创建的指令应用到应用程序中。基于编译器的工具确切地知道 C 操作的组合与每一条新指令的相关性，因此当发现新指令有助于提高性能和提高代码密度时，就可以对该指令进行实例化。此外，一旦指令集被冻结，SoC 已经创建完成，编译器将保留 C 代码和指令的相关性知识。即使在 C 源程序发生改变之后，编译器也能够使用相同的扩展指令。

第四，自动发生器能够做出比人类技术人员更好的指令集扩展决策。发生器不会受技术人员偏见的影响，这些影响会不利于创建新的指令（设计惯性）；也不会受一些传言的影响，比如某些指令的性能更好；它对门数和执行周期具有完全的、准确的估计，能够进行全面系统的成本/利益分析。这种利益组合能够同时满足专用处理器的两个承诺：更低的价格；最优化芯片的开发速度更快，并且一旦根据不断发展的系统需求创建了芯片，更容易对芯片进行重编程。

## 14.2 Tensilica Xtensa 体系结构概述

在嵌入式系统的设计中，一直依赖处理器，且其起着非常重要的作用，但是这些系统在性能和复杂性方面的根本变化正在改变着处理器所起的作用。在每个嵌入式应用之间（包括数据通信、电话、图像和消费类系统），算法和协议变得越来越多样化和复杂，数据类型也越来越丰富，在这些新型的系统中，需要新型的处理器。这样的要求是很荒谬的，即每一个新的处理器必须完全支持特定的新算法、数据类型和目标应用的带宽，而同时与传统处理器相比，又必须做到体积更小、功耗更低（见图 14-20）。

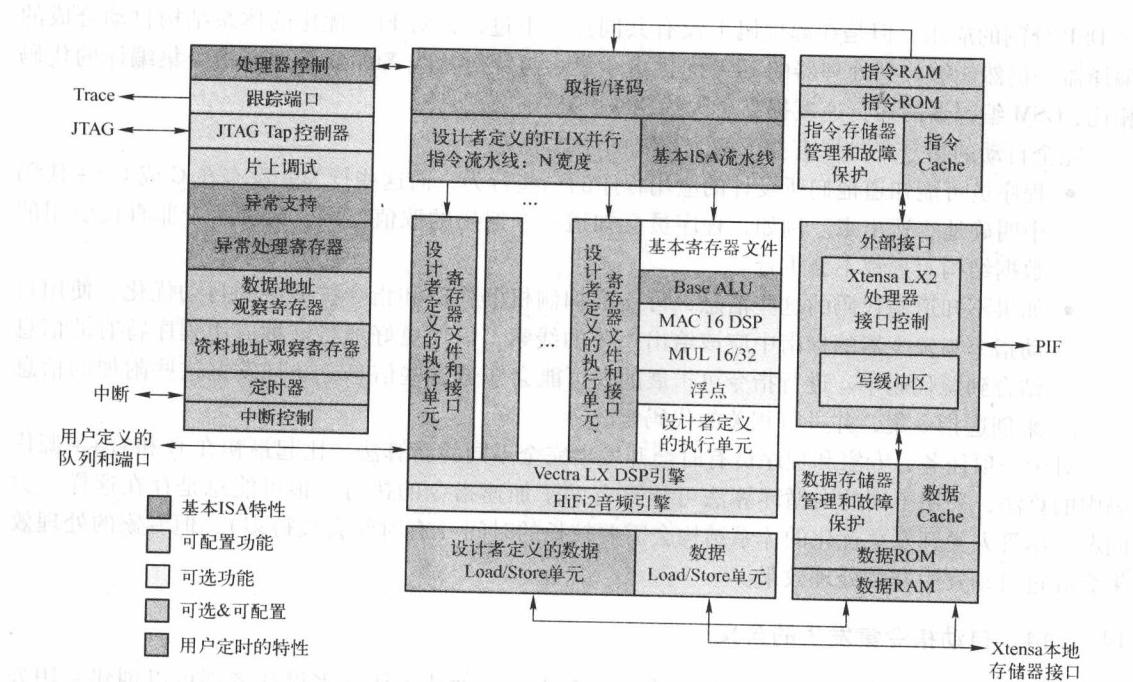


图 14-20 Tensilica Xtensa LX2 体系结构框图（经泰思立达公司许可使用）

要解决这个矛盾，需要有一种新的专用处理器的处理器自动生成方法。针对这个目的，Tensilica 已经设计了 Xtensa Xplorer 设计环境和 Xtensa 处理器发生器。Xtensa 处理器发生器结构框图如图 14-21 所示，通过快速描述处理器需要的关键指令、存储器和外设以及接口功能，芯

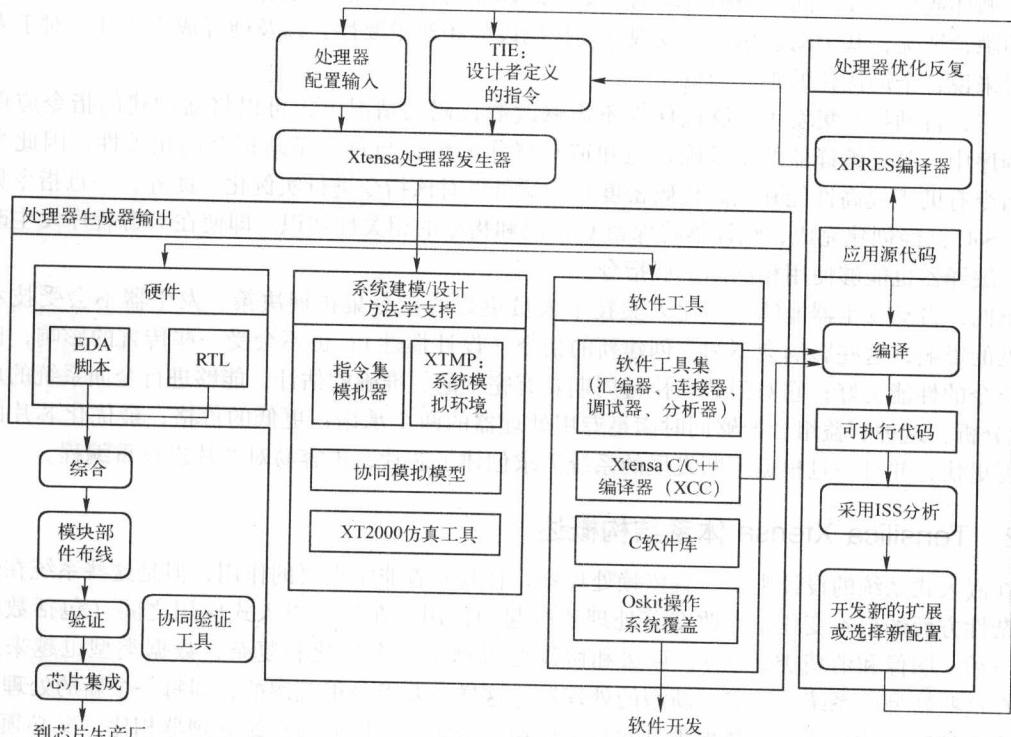


图 14-21 Xtensa LX2 核框图（经泰思立达公司许可使用）

片设计者能够快速发掘另外的设计方法。Xtensa 处理器发生器为定制处理器的快速设计给出了一个完整的处理器硬件设计、验证和软件开发环境。与通用嵌入式处理器相比，该发生器所设计出的处理器具有执行应用程序速度快、功耗低、代码少的优点。

这些处理器的基础是 Xtensa 指令集体系结构。对于所有的 Xtensa 处理器来说，基本的体系结构都是相同的。这种指令集的设计目的是为了完全满足高性能、大容量嵌入式通信和消费类应用程序的独特需求。Xtensa 体系结构有如下 4 个主要设计目标：

- 代码尺寸小；
- 改进通用嵌入式处理器性能；
- 低功耗；
- 对新的应用程序的需求提供无缝扩展支持。

此外，自从 15 年前 RISC 体系结构出现以来，因为相关技术在 3 个方面的发展，现在可以创建一种新的体系结构基础：

- 半导体技术从  $0.15\mu\text{m}$  到  $65\text{nm}$ ，提供了实现基础；
- 从桌面的通用计算到嵌入式系统以数据为中心的计算的转换，是最主要的应用领域；
- 在体系结构原理、编译技术和编程语言以及技术方面的不断进步；
- 移动消费类应用对低功耗的需求。

### 14.3 指令集设计原则

指令集的设计是非常有艺术性的。许多指令集特性并不是独有的，但是可以将这些特性以独特新颖的方式组织，以改善指令集结构；尤其是当针对新应用领域对指令集进行优化时，将会极大地改善结果。

指令集体系结构 (ISA) 设计需要在多个设计目标之间进行平衡，这些设计目标包括：

- 实现各种算法的机器代码大小；
- ISA 对新算法和应用的可扩展性和可适应性；
- 采用这种 ISA 的处理器执行一些算法的性能；
- 采用这种 ISA 的处理器执行一些算法的功耗；
- 采用这种 ISA 的处理器价格；
- 体系结构对未来处理器实现的适宜性；
- 采用这种 ISA 的处理器的设计复杂性；
- 以该 ISA 为目标的高级程序语言编译适宜性。

指令集体系结构对于处理器的性能有一方面的直接影响和两方面的间接影响。ISA 直接决定了实现给定算法所需要的指令数量。处理器性能的其他方面包括最小的时钟周期和每条指令的平均时钟周期数，这是指令集实现的主要属性，而指令集的特点也影响了设计者同时满足时钟周期长度和每条指令的时钟数两个目标的能力。例如，一些编码方式的选择可能会要求指令执行过程中增加一些附加的逻辑，此时设计者会有两种做法，或者增加时钟周期的长度，或者增加流水线的级数，而这样做会增加每条指令执行的时钟周期数（指令延迟）。

### 14.4 Tensilica Xtensa 处理器的独有特性

RISC（精简指令集计算）处理器的设计理念出现于 20 世纪 80 年代。RISC ISA 的使用使得处理器设计者能够极大地降低处理器每条指令的周期数和时钟周期长度，而同时又不会使程序的指令数量过度增加。RISC ISA 提高了处理器性能，降低了设计复杂性，在给定性能的基础上降低处理器实现的成本，并且非常适合高级编程语言的编译。

奇怪的是，对于术语 RISC 至今还没有一个全面或者令人满意的定义，但是 RISC 处理器通常具有如下特性：

- 固定的指令字长；
- 大量寄存器文件对计算操作的支持；
- 简单并固定的指令域编码；
- 存储器的访问通过寄存器的 Load 和 Store 进行；
- 存储器寻址方式很少（通常只有一个，最多不会超过 4 个）；
- 消除了那些不利于指令流水线执行的特点（不同的指令延迟和微编码指令）。

另一方面，大部分 RISC ISA 都用于高性能的桌面计算环境，这样的环境中的磁盘存储量比较大。它们并没有以紧凑的机器语言代码为优化目标，特别是与非 RISC ISA 相比，在对应用程序编码时，RISC 指令集通常需要更多的程序量。在当今的许多嵌入式应用程序中，代码存储量的成本（片上 RAM/ROM）通常要大于处理器（门数）的成本，因此 RISC 处理器的使用有时对于成本比较敏感的应用来说是受限制的。如果将 RISC 的优势和紧凑的代码尺寸相结合，将非常适用于嵌入式应用，这种结合正是 Xtensa ISA 开发人员所面对的主题之一。

Xtensa 指令集既利用了已有 RISC 体系结构的优势，又加入了新的指令集体体系结构思想。Xtensa ISA 的大部分特性属于 RISC，但是同时又兼有早期的 CISC 体系结构所具有的优势，例如紧凑的程序代码。

Xtensa 内核 ISA 是一组 24 位的指令，执行 32 位的操作。在选择指令字长度时要首先考虑的是代码长度，指令本身的选择是根据广大嵌入式应用程序的需要来进行的。内核 ISA 拥有许多优秀的特点，例如复合操作的指令提高了它对于嵌入式应用的适用性，而同时又避免为了满足一些应用程序的需要而以牺牲其他应用程序的成本或功耗为代价（例如，对额外寄存器文件端口的需要）。这样的属性在 Xtensa 体系结构中可以选择性的实现，也可以根据特定应用领域的需要设置协处理器。

Xtensa ISA 的组织方式是，一组核心指令，再加上各种针对特定应用领域进行扩展的可选择功能包。这样设计者可以在处理器核中只选择需要的功能，提高解决方案的效率。核心 ISA 中提供了普通控制应用程序所需要的功能，可以自由选择对位或字节进行操作。内核还对第三方软件进行支持，因此不允许将这些功能从内核中删除。另一方面，数字计算应用，例如数字信号处理器，可以通过可选的 ISA 功能包对特定的应用领域实现非常好的支持，例如数字滤波器可以选择 MAC16，对于高端音频处理应用可以选择浮点协处理器。

基本的 Xtensa 体系结构基本是按照 RISC 的思想来构建的，但同时又引入了一些新的技术来改善程序的指令数量和每条指令的平均长度，这些技术对于提高性能和降低成本是非常有帮助的。Xtensa ISA 的最初设计理念是，必须基于 RISC 思想，实现很好的代码密度、固定长度、高性能编码方式，包括通用寄存器文件和 load/store 体系结构。

为了获得满意的代码密度，Xtensa 处理器增加了一种简单的变长编码方式，这种方式的采样不会降低性能。Xtensa 体系结构通过对一些功能的平衡，进一步优化了处理器实现的成本，例如寄存器文件、控制流操作、数字逻辑指令、load/store 功能，这些都有利于现代嵌入式软件中经常出现的操作，并且适用于小而快速的现代深亚微米实现。

## 14.5 寄存器

为了维护性能，RISC 指令集必须支持至少两个源寄存器文件域和一个目的寄存器文件域。如果只针对代码密度进行优化，一般的寄存器指令大约只有 2 个寄存器域，一个作为源寄存器，另外一个既作为源寄存器也作为目的寄存器。这种设计方法有时会降低代码长度，但是却会提高程

序的指令数量；对于寄存器数量较少的指令集，使用的寄存器域也少，并降低了指令的长度。同时，这样的指令集需要内存中有更多的变量和临时值，需要更多的load和store指令，并且提高了程序中的指令数量。因此这样的设计方法既提高了程序执行的周期数量，也增加了功耗。

随着寄存器数量的增加，两个操作数的指令格式优势越来越小。特别的，要达到很好的RISC性能，通常至少需要16个通用寄存器。3个4位的寄存器域至少需要12位来进行编码，此外还需要操作码和常数域。因此16位的编码作为有些处理器所采用的编码方式，对于提高性能来说是不够的。

## 14.6 指令长度

先前的RISC体系结构在代码长度和性能之间没有达到很好的平衡，因为RISC ISA设计者仅限于一些指令长度为16位和32位。当然，使用与处理器的数据字长成简单比例的指令字是有优势的，但是有时候打破这种限制会获得更多好处。

Xtensa处理器使用的24位定长编码（见图14-22）作为起点，24位的长度足以获得高性能；而同时对功能强大的指令也有足够的可扩展性，可以减小程序中的指令数量。与普通的RISC32位指令字相比，Xtensa ISA的24位编码方式减小了25%的指令长度，比大部分32位RISC指令集降低了代码大小；最重要的是，24位非常易于适应采用32位数据通路长度的处理器。

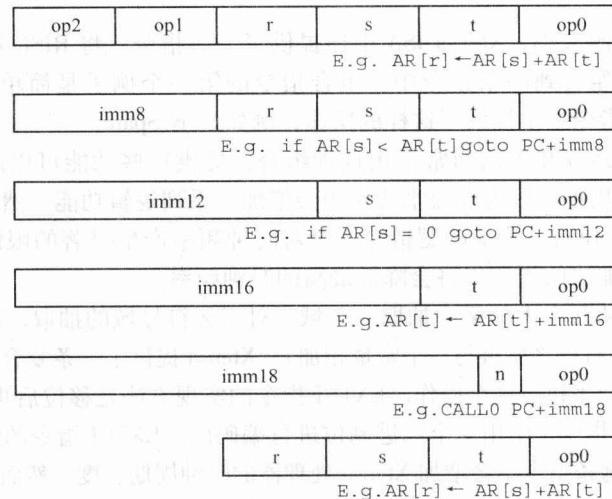


图14-22 Xtensa典型的16位/24位指令类型（经泰思立达公司许可使用）

Xtensa体系结构采用4位的寄存器域，如图14-23所示，是获得高性能的最小长度，也是适应24位指令字的最大长度。

与8位和16位通用寄存器之间的差别相比，16位和32位通用寄存器之间的性能差别很小（约5%），其他的特性很容易对这个差别进行弥补（例如复合指令和寄存器窗口）。程序中指令数量的提高（约5%）完全可以由24位和32位编码方式之间的差别来弥补（约25%）。

许多采用5位寄存器域的指令集并不提供32个通用寄存器。大多数处理器专门设置一个寄存器的值为0，尽管增加几个额外指令操作码可以很容易消除零值寄存器的需求（例如Xtensa的NEG指令）。同样，其他寄存器也有专门的用途，这也可以通过在指令集中包含其他特性而避免。例如，一些RISC体系结构将其31个通用寄存器中的2个专门用于异常处理，还有一个寄存器专门用于全局的区域指针。这样，实际上体系结构只给程序员提供了28个通用寄存器，用于变量和临时存储；与使用4位寄存器域的指令集相比，也就多出12个通用寄存器。

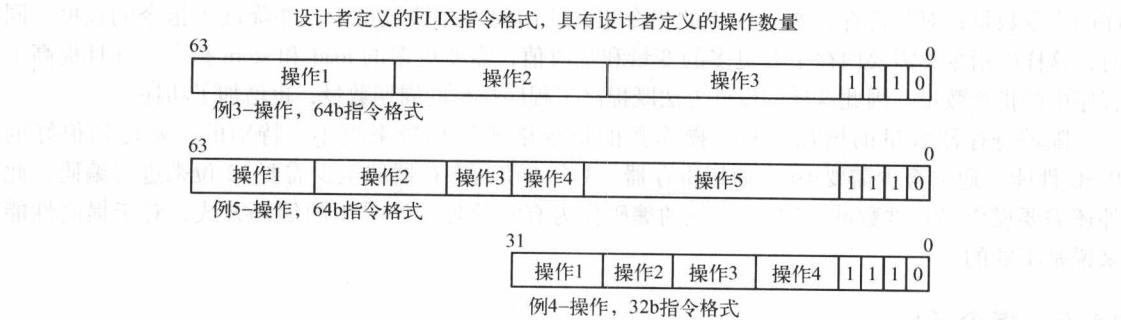


图 14-23 32位/64位扩展指令（经泰思立达公司许可使用）

一种普遍的做法是，将通用寄存器划分为由软件使用的调用者和被调用者保存，进一步限制了更大寄存器文件的使用。Xtensa ISA 没有使用这种方法，这样使得 16 个寄存器的使用效果与 32 个寄存器的效果大致相同。Xtensa ISA 表明，全功能 RISC 指令集的 24 位编码方式是完全可能的。此外，Xtensa ISA 通过扩展指令长度支持并行操作，如图 14-23 所示。Xtensa ISA 是处理器设计的一个很大的进步。

## 14.7 复合指令

为了改善性能和代码大小，Xtensa ISA 中还提供了一些指令，将 RISC 和其他处理器指令集中常见的多种指令功能组合到一条指令中。组合指令的第一个例子是简单的“左移和加法/减法”，高端体系结构的指令集中提供了这样的操作，例如 SUN Sparc。

小常数的地址算术运算和乘法通常采用这种组合，提供这些功能可以降低指令数量，而同时会增加处理器的时钟周期，因为在流水线站中会增加一系列逻辑功能。然而，各种实现已经表明，当移位限制在 0~3 位时，增加的逻辑并不会对时钟频率产生显著的限制。因此，采用相关 ISA 并提供任意移位和加法的指令集将会降低最高的时钟频率。

右移操作通常用于从几个大的字中抽取一个域。对于无符号域的抽取，通常要使用两条指令（左移之后再右移，或者右移之后再与一个常量相加）。Xtensa 提供了一条复合指令，即 EXTUI（抽取无符号立即数）指令，来执行这个操作。EXTUI 指令的实现方法是移位后再与指定的 mask 进行加法操作，该 mask 是在指令字中用 4 个二进制位进行编码的。EXTUI 指令的逻辑加法部分是非常简单的，在 ISA 中包含这条指令不会增加 Xtensa 处理器的时钟周期长度。然而对于有符号域的抽取指令，情况就不是如此了，因此在 Xtensa ISA 中并不包含相关的有符号 EXTSI 指令。

## 14.8 分支

无论是 RISC 还是其他体系结构，大部分处理器指令集（例如 ARM、Intel Viiv、Freescale PowerPC 和 Sun SPARC）都有一条设置条件码的比较指令，随后会有一条条件分支指令来测试条件码，用于程序流的控制。在大部分 RISC 指令集中，条件分支构成了 10%~20% 的指令，通常每一条条件分支指令都与一条比较指令一起使用，这种风格的指令集是非常浪费的。一些指令集（例如 ARM、MIPS、HP PA-RISC、Sun SPARC V9）提供了比较和分支复合指令，具有很强的灵活性。Xtensa ISA 提供了非常有用的比较分支复合指令。

选择这样的组合时，需要在每一条比较和分支指令的用处与所占用的操作码空间之间进行平衡，尤其是当所针对的指令是 24 位（与 32 位不同）指令编码的指令集时。其他指令集没有进行测试，与其他比较分支复合指令分开的指令集相比，比较和分支复合指令减小了指令的数量，即使与部分功能比较分支复合指令相比，结果也是如此。一些 Xtensa 处理器可能需要通过提高每条指令的

时钟周期数来实现一些比较和分支复合指令，但是这些复合指令的总体性能提高还是绝对的。

Xtensa ISA 的比较分支复合指令还支持立即数的比较，并使用常数的编码来增加这些指令的用途。BEQI、BNEI、BLTI 和 BGEI 指令还使用一个 4 位的域，对各种普通常数进行编码。BLTUI 和 BGEUI 指令使用不同的编码方式，因为无符号比较的数值范围是不同的。

Xtensa 处理器的比较和分支复合指令集将所有的这些立即数压缩为一个指令字，这样域的大小就会降低。这些指令将在一个 24 位的指令字中，集合了比较操作码、两个源寄存器域和一个 8 位的 PC 相对偏移量。在一些情况下，8 位的相对偏移量显得太小了，一些编译器或汇编器通过使用范围较大的无条件分支指令进行补充。Xtensa ISA 还提供了一系列比较和分支复合指令来测试零值，这是很常用的情形。这些比较和分支复合指令具有一个 12 位的 PC 相对偏移量，这样范围会更大。

Xtensa 体系结构在指令集设计中加入了另外一个重要并独特的目标——对可扩展性的完全支持，可以支持在新的指令中和紧耦合的协处理器中使用的新的数据类型（见图 14-24）。Xtensa ISA 还使用另外的方法来支持协处理器的条件分支，还支持增加 16 个 1 位的布尔寄存器。Xtensa ISA 的 BF（条件假分支）和 BT（条件真分支）指令测试这些布尔寄存器，并根据结构进行分支。

设计者定义的协处理器可以实现一些指令，根据对它们所支持的数据类型进行比较，来设置布尔寄存器。

所有 Xtensa 协处理器与 BF、BT 指令共享基本 ISA 的布尔寄存器集，这种方式可以充分利用 Xtensa ISA 的 24 位短

指令字，这种方法是其他许多早期处理器 ISA 中比较和分支条件码的另外一种新实现。使用单位比较结果寄存器（Xtensa、MIPS），而不使用多位比较结果寄存器（大部分其他 ISA），增加了比较操作码的数量，而同时降低了分支操作码的数量。这种 ISA 设计方法引入了大量专用户易于实现的分支和条件操作，这对于专门针对可扩展性设计的 ISA 是非常重要的。

Xtensa ISA 还提供了一种通用、零开销循环特性，类似于一些 DSP（数字信号处理器）中的循环特性。大部分 RISC 处理器使用已有的条件分支指令来实现软件循环，然而，这种方式虽然可以减少操作码的大小，但是会增加程序执行的周期数量，进而降低执行的速度。对于许多 RISC ISA 来说，循环开销包含加法、比较和条件分支 3 种指令。当循环体较小时，分支开销对性能的影响是很高的。对于小的软件循环，许多编译器采用一种称为循环展开（见图 14-25）的优化技术，在两个和更多个循环反复之间分摊循环开销。但是这种方法会重复循环体，极大地增加代码大小。

与之相反，许多 DSP 和一些通用处理器提供了执行类似循环的其他方法。第一种方法是提供一条指令，以固定的次数重复后面的指令（例如 TI TMS320C2x、Intel x86）。对于指令循环，重复的前缀指令消除了循环开销，并且由于不需在一个循环中反复取同一条指令，还可以节约功耗。一些具有重复指令的 ISA 规定，处理器在运行循环的过程中不能被打断。

这种限制可能会造成不可接受的中断延迟，因为一些循环的执行可能需要许多个机器周期。对简单重复前缀指令处理的一种改进就是，在降低或消除循环开销的前提下，多次重复一个指令块（例如 TI TMS320C5x）。Xtensa ISA 提供了 LOOP、LOOPGTZ 和 LOOPNEZ 指令，能够实现零循环开销。Xtensa ISA 的 LOOP 指令，消除了增加循环指数和比较以及分支操作所需要的指令执行周期，还避免了 taken-branch 消耗（这种消耗通常是由于基于条件分支指令对循环进行编译而产生的）。

Xtensa ISA 证明了如何能够在通用处理器 ISA（而不是 DSP）中降低循环消耗，从而既提高

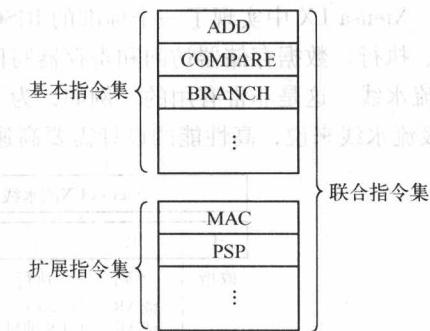


图 14-24 扩展指令集

L1	Ins A	Ins A
	Ins B	Ins B
	Ins C	Ins A
	IF X0 S3	Ins B
	then L1	Ins A
		Ins B

图 14-25 展开的循环

执行性能，又减小代码大小。Xtensa 体系结构对通用分支指令的重要贡献包括如下 6 个：

- 可以在 RISC ISA 中选择比较效率最高的比较和分支指令；
- 根据编码的立即数执行比较和分支，包括按位分支指令；
- 针对一般情况（测试零）加长目标 specifier 的指令格式；
- 在一个 24 位的指令字中编码所有分支指令；
- 对协处理器布尔寄存器（条件码）分支可扩展的支持，同时可以进行布尔逻辑操作；
- 零开销循环，消除了分支执行延迟，并减小了代码大小。

## 14.9 指令流水线

Xtensa LX 中实现了一个标准的 RISC 5 级流水线，如图 14-26 所示，其中包括取指、寄存访问、执行、数据存储器访问和寄存器写回。此外，为了增加设计灵活性，Xtensa LX 还支持 7 级的流水线。这是非常有用的，例如，为了节约功耗和芯片面积，有时需要使用低速存储器。对于 5 级流水线来说，高性能的设计需要高速的存储器。

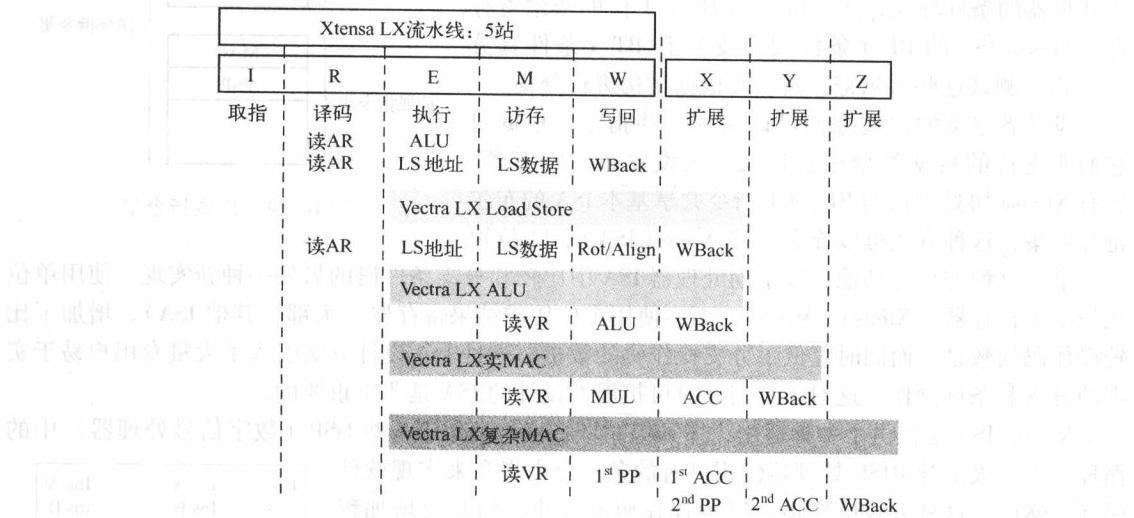


图 14-26 Tensilica Xtensa LX2 5 级流水线（经泰思立达公司许可使用）

## 14.10 有限的指令常数宽度

所有 Xtensa 基本指令都不超过 24 位，因此指令字中的常数域宽度受到限制。Xtensa 体系结构采用了多种途径来解决这个问题。Xtensa ISA 使用小的常数域来满足大多数普通常数的需求，Xtensa 指令对常数进行编码，而不是直接表达。对每一种指令类型进行统计学分析，可以找出 N（例如 16）个最常用的常量，作为常数的编码制。

Xtensa 在 ADDI4 指令中使用了这种技术，16 个值的选择为 -1, 1 ~ 15，而不是 0 ~ 15。加 0 操作是没有意义的（有 MOVE 指令实现），比较常见的是加 1。在位逻辑操作（例如 AND、OR、XOR）中所使用的常数，要表示各种类型的位屏蔽，通常不适用于小的常数域。一串 0 后面跟一串 1，以及一串 1 后面跟一串 0，这样的位模式是很常见的。基于这个原因，Xtensa 体系结构中有些指令就没有直接将屏蔽值放入指令字中。EXTUI 指令（在前面已经描述过）会在一串 0 后面跟一串 1 的屏蔽之前执行一次移位，1 的位数就是指令中的常数域。

Xtensa 的 load/store 指令格式中有一个 8 位的常数偏移量与来自一个寄存器的地址相加，Xtensa ISA 既充分利用了这 8 个二进制位，也为 8 位不够的情况提供了一种简单的扩展方法。

Xtensa 的 load/store 偏移量是零扩展的，而不是符号位扩展的，因为 128 ~ 255 之间的值是 load/store 指令中最常用的，-128 ~ -1 之间的值不常用；并且，偏移量是左移的，大部分参考值都是从对齐的基寄存器到对齐的地址映射。

32 位的 load 和 store 偏移量要移动 2 位，而 16 位的 load 和 store 偏移量要移动 1 位，而 8 位的 load 和 store 偏移量不移动。大多数 load 和 store 是 32 位的，因此这种技术还预留了 2 个二进制位。如果在 load/store 指令（或者 ADDI 指令）中出现的 8 位的常数偏移量不够，Xtensa ISA 还提供了 ADDMI 指令，该指令将 8 位常数左移 8 位，这样两条指令就有了 16 位的范围，8 位来自 ADDMI，8 位来自 load/store 或者 ADDI 指令。

## 14.11 短指令格式

Xtensa ISA 包含一个核心指令集，在指令集的所有实现中都必须实现该指令集；还包含一个可选指令包，在指令集的实现中可以有选择地对这些指令进行实现。最常用的指令包之一就是短指令格式包。该指令包通过减小指令字的平均长度，进一步缩小代码大小（见图 14-27）。当使用这些短格式指令时，Xtensa 从固定长度（24 位）指令集转换到具有两种指令大小（24 位和 16 位）的模式。Xtensa 体系结构并没有采用其他 RISC 处理器的方法，采用模式转换将 16 位指令加入到 ISA 中。

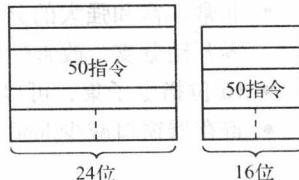


图 14-27 指令宽度

Xtensa ISA 的 24 位和 16 位指令格式是同时运行的，因此，从一种指令格式转换为另外一种指令格式时不会引入开销，因为 Xtensa 端指令格式是可选的，这些模式仅仅用来压缩代码的大小，在 Xtensa ISA 的 16 位指令中并没有增加新的功能。能够编码为 16 位的指令集包含了最常用的指令，在大多数指令集中，最常用的指令有 load、store、分支、加法和移动，这些也正是出现在 Xtensa ISA 的 16 位指令集中的指令。

只有最常用的指令需要压缩编码，因此仍然有 3 个寄存器域（因为操作码域很小），窄小的、编码的常数域能够满足大部分应用需求。16 位指令中有 16 个操作码，构成应用程序中大约 1/2 的 Xtensa 指令能够用其中的 6 个进行编码，其余 3 个 4 位的域保留给寄存器地址或常量。

## 14.12 寄存器窗口

Xtensa 体系结构的另外一个重要特点是窗口化的寄存器文件，如图 14-28 所示。寄存器窗口减小了代码大小，还提高了性能。其他处理器中也实现了寄存器窗口，如 Sun 的 SPARC ISA。“寄存器窗口”的名称描述了一种典型的实现方式，指令中的寄存器域描述了处于当前窗口中的一个寄存器，该窗口位于更大的寄存器文件中。寄存器窗口的使用，避免了在进程进入和退出时，进行寄存器的保存和恢复。设计了寄存器窗口的处理器不用将寄存器的值保存到堆栈中并从堆栈恢复，只需要修改寄存器偏移量指针，该指针会将一些寄存器隐藏，而将另外一些向寄存器提供使用。

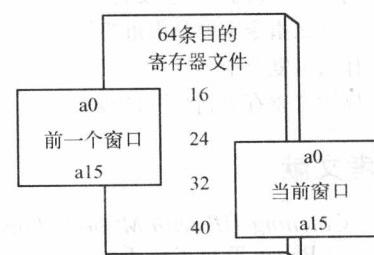


图 14-28 寄存器窗口  
(经泰思立达公司许可使用)

可供使用的寄存器通常不包含有效的数据，可以直接使用。调用者和被调用者的寄存器窗口，在物理寄存器文件中会发生重叠，这样当参数通过寄存器传输时，也避免了参数混乱。最后，寄存器窗口使得将变量或临时值分配到寄存器中，没有任何开销。鼓励使用寄存器，比使用内存单元的速度要快。

与 SPARC 的固定窗口重叠增量不同, Xtensa ISA 采用了一种变量增量应用于寄存器窗口。这种方法允许使用更小的物理寄存器文件, 因此实现成本较低。例如, 许多 Sun SPARC ISA 都使用了 136 个人口的物理寄存器文件, 而 Xtensa ISA 实现的寄存器文件只有 64 个人口, 却能获得相同的性能。Xtensa ISA 给出了新的方法, 来测试窗口溢出和下溢, 并组织堆栈格式。

### 14.13 Xtensa L2 总结

Xtensa 体系结构为嵌入式处理器体系结构做出了许多重要的贡献, 如下:

- 一个完全的、被划分为 16 个可见窗口的寄存器文件, 3 种操作数编程模式, 最多的 32 位指令编码, 有助于改善性能、一般性和代码大小;
- 将经常出现的指令组合称为复合指令;
- 对常用立即值进行编码, 以改善性能和代码大小;
- 非常丰富和强大的分支结构, 包括比较和分支、位测试分支、协处理器条件码和分支及零开销分支, 改善性能和代码大小;
- 16 位指令子集, 可以与 24 位基本指令自由混合使用, 进一步压缩了代码大小;
- 寄存器窗口减少 load/store 操作的数据交互, 进一步提高了性能和代码密度。

## 习题

1. 当前 SoC 设计的意义是什么?

2. 设计复杂性和设计生产力的比较有哪些变化?

3. 列出芯片设计者最关注的两个问题。

4. 列出半导体集成度提高的 3 个好处。

5. 系统开发人员所面对的 2 个关键挑战是什么?

6. 什么是“可编程开销”?

7. SoC 开发的主要目标是什么?

8. 可配置、可扩展处理器是如何产生的?

9. 列出 SoC 设计流程的关键要素。

10. 列出自动指令集生成的 4 种含义。

11. 列出 Xtensa 体系结构的 4 个主要目的。

12. 列出 ISA 中需要平衡的 8 个目标。

13. 列出 RISC 的 6 个主要特性。

14. 为什么指令宽度是很重要的?

15. 什么是复合指令?

16. 描述“寄存器窗口”的概念。

## 参考文献

*Catching UP with Moore's law: How to fully exploit the benefits of nanometer silicon.* White Paper. Tensilica, Inc.

Dixit, Ashish. 2004. *Introducing the Xtensa LX Processor Generator.* Tensilica Inc.

Leibson, Steve. 2003. *Designing with configurable processors instead of RTL.* Tensilica Inc.

Rowen, Chris. 2002. *Reducing SoC simulation and development time.* Computer, December 2002, reprint.

Rowen, Chris. 2006. *The reinvention of the microprocessor.* Tensilica Inc.

*Xtensa architecture and performance.* White Paper. October 2005. Tensilica, Inc.

*Xtensa LX microprocessor overview handbook.* Data Book. Tensilica Inc.

# 数字信号处理器

- 本章目标：理解 DSP 控制器的体系结构

- 学习内容：

读者将学习到下列处理器的基本体系结构和特性：

1. Texas Instruments 公司的 TMS320C55x。
2. Analog Devices 公司的 ADSP-BF535 (Blackfin)。

## 15.0 DSP 概述

与通用处理器相比，专用的数字信号处理器器件可以极大地提高数字信号处理器（DSP）算法的性能。在本书前面的内容中我们已经看到，COTS 和 IP 核可能都有 DSP 方面的扩展，但是从本质上来说它们还是通用器件。对于以 DSP 功能为主的应用来说，则要专门设计器件，以满足它们的需要。

### 15.1 TMS320C55x

TMS320C55x (C55x) 专门用于个人和移动处理应用以及数字通信基础设施，体系结构如图 15-1 所示，在成本和功耗方面都有优势。与 120 MHz 的 C54x 相比，300 MHz 的 C55x 的性能提高了大约 5 倍，而功耗只有 C54x 的 1/6。

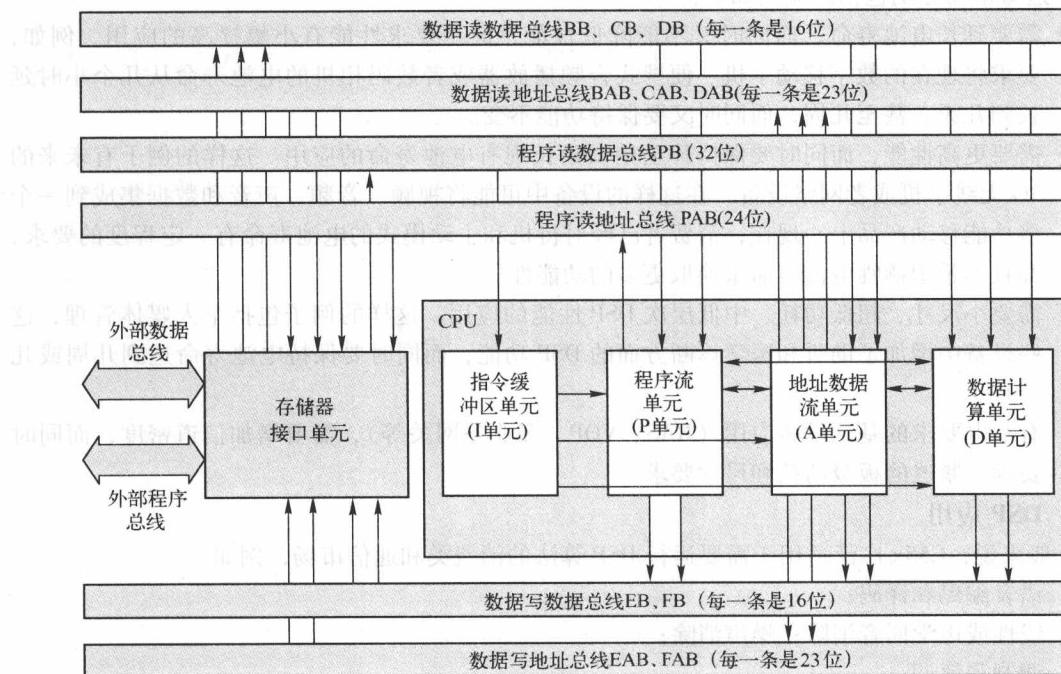


图 15-1 TMS320C55x CPU 体系结构（经德州仪器许可使用）

C55x 内核通过低功耗设计和先进的电源管理技术，实现了超低功耗。C55x 具有高级的节电可配置性，并且还具有自动切换的电源管理，这些都是对用户透明的。

C55x 内核采用了双 MAC（乘累加）体系结构，具有并行指令、加法累加器、ALU 和数据寄存器等器件，它的周期效率是 C54x 的两倍。C55x 实现了一种高级指令集，该指令集是 C54x 的一个超子集，结合了扩展的总线结构，还支持加法硬件执行单元。

C55x 指令集是针对增加代码密度而设计的，可以降低系统成本。C55x 指令集中的指令字长可变，范围为 8~48 位。由于采用了这种可变的指令字长度，C55x 降低了每个功能的控制码大小，从而提高了存储密度。控制码大小降低意味着对内存的需求降低，从而会降低系统成本。

### 15.1.1 TMS320C55x 的特性

TMS320C55x 综合了超低功耗性能和低系统成本特点，以满足个人和便携式应用不断数字化和小型化的需求。C55x 体系结构和硬件设计有如下 3 个相互关联的目标。

#### (1) 超低功耗

允许提高便携式设备的电池寿命，提高节电基础设施系统的信道密度。

#### (2) 高效的 DSP 性能

可以在系统中增加更多的功能，或者提高现有算法的处理速度。

#### (3) 高代码密度

对于给定的功能，要降低内存的需求，进而降低系统成本和/或缩小系统尺寸。所有这些都取决于更加紧凑的编码方式。

## 1. 市场分割

C55x 支持 4 类基本的市场分割，根据不同的侧重点，满足低功耗、低系统成本和高性能的需求。这些市场分割包括的应用如下：

- 需要延长电池寿命，而同时又不能降低性能，甚至要求性能有小幅提高的应用。例如，要求将现在的数字移动手机、便携式音频播放器或者数码相机的电池寿命从几个小时延长到几天，甚至几周，而同时又要保持功能不变。
- 需要更高性能，而同时要保持甚至稍微延长现有电池寿命的应用。这样的例子有未来的 3G 无线手机或者网络设备，在这样的设备中可能将视频、音频、声音和数据集成到一个多功能移动产品上。现在，消费者已经对待机和主动模式的电池寿命有一定程度的要求，并且不希望牺牲电池寿命来换取更多的功能性。
- 需要小尺寸、超低功耗、中低层次 DSP 性能的应用。这样的例子包括个人媒体管理，这些设备中增加了助听和医学诊断方面的 DSP 功能，而同时要保持电池寿命达到几周或几个月。
- 有节电要求的基础设施应用（RAS、VOP、多业务网关等），需要增加信道密度，而同时要满足紧迫的板级功耗和尺寸要求。

## 2. DSP 应用

一般来说，C55x 广泛适用于需要运行 DSP 算法的消费类和通信市场，例如：

- 语音编码和译码；
- 线性或声学回音消除；噪声消除；
- 调制和解调；
- 图像、音频压缩和解压缩；
- 语音加密、解密；

- 语音识别、语音合成。

### 15.1.2 C55x 的主要特征

C55x 中具有非常丰富的特征，具有处理效率高、低功耗和易于使用等特点。这些重要的特征包括如下几项：

- 一个 32 位  $\times$  16 位的指令缓冲队列。
- 两个 17 位  $\times$  17 位的 MAC 单元。
- 一个 40 位的 ALU。
- 一个 40 位的筒形移位器。
- 一个 16 位的 ALU。
- 4 个 40 位的累加器。
- 12 条独立的总线：
  - 3 条数据读总线；
  - 2 条数据写总线；
  - 5 条数据地址总线；
  - 1 条程序读总线；
  - 1 条程序地址总线。
- 用户可配置的 IDLE 域。
- 变长指令缓冲，并实现高效块重复操作。
- 在一个周期内执行双 MAC 操作。
- 执行高精度算术和逻辑操作。
- 可以将 40 位的结果最多向左移动 31 位，或向右移动 32 位。
- 可以与主 ALU 并行执行简单的算术运算。
- 保存计算结果，减少访存次数。
- 利用 C55x 的并行性，并行给各种计算单元提供要处理的指令和操作数。
- 改进了低功耗管理的灵活性。

### 15.1.3 指令集体系结构

C55x 体系结构通过提高并行性，并致力于降低功耗，获得了很好的低功耗性能。

其中的 CPU 支持一种内部总线结构（见图 15-2），由如下总线构成：

- 1 条程序总线；
- 3 条数据读总线；
- 2 条数据写总线；
- 专用于外设和 DMA 操作的总线。

对这些总线进行设置，可以使 CPU 在一个周期内最多执行 3 次读数据和 2 次写数据操作。DMA 控制器可以在每个周期执行最多 2 次数据传输操作，这些操作都是与 CPU 相互独立并行进行的。

#### 1. 指令流水线

C55x DSP 将流水线划分为 7 级，分别进行取指、译码和执行操作，如图 15-3a 和 15-3b 所示。7 个站分别为：

- 取指站，将程序从内存读取到指令缓冲队列中；

- 译码站，对指令进行译码，并将任务递交给其他主要功能单元；
- 地址站，计算数据的地址，在程序跳转的情况下还要计算分支地址；
- 访存1站/访存2站，将数据的地址发送给存储器；
- 读取站，将操作数传送到B总线和C总线和D总线；
- 执行站，执行A单元和D单元的操作，并向E总线和F总线写数据。

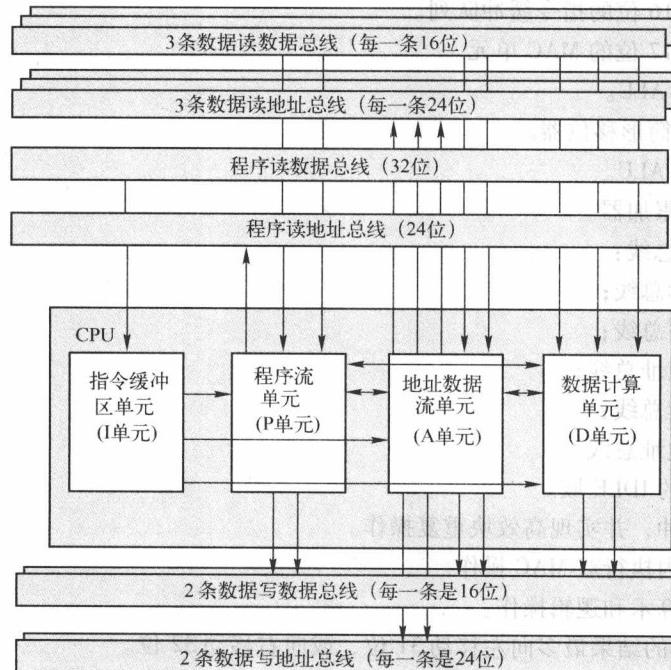


图 15-2 TMS320C55x 器件总线 (经德州仪器许可使用)

流水线站				时间
PF1	PF2	F	PD	

流水线站	描述
PF1	将程序地址发送给存储器
PF2	等待存储器的响应
F	从存储器中读取一条指令，并将它放入IBQ中
PD	对IBQ中的指令进行预译码（识别指令的起始和结束位；识别并行指令）

a) TMS320C55x 流水线前面的站点

译码 (D)	地址 (AD)	Access1 (AC1)	Access2 (AC2)	读 (R)	执行 (X)	写 (W)	写+ (W+)
-----------	------------	------------------	------------------	----------	-----------	----------	------------

注： 仅对于存储器写操作

b) TMS320C55x 流水线后面的站点

图 15-3 (经德州仪器许可使用)

C55x 流水线是受保护的，这样就意味着在出现总线冲突时，可以自动插入周期。在如下情况下会插入流水线保护周期：

- 一条指令要向一个单元写入数据，但是前面的指令还没有完成对该单元的读操作（插入一个多余的周期，让读操作先执行）；
- 一条指令要从一个单元读取数据，但是前面的指令还没有完成对该单元的写操作（插入一个多余的周期，让读操作先执行）。

## 2. CPU 特性

C55x CPU 中有两个乘累加 (MAC) 单元，每一个单元都可以在一个周期内执行一次 17 位  $\times$  17 位的乘法操作。40 位的主算术逻辑单元 (ALU) 由 16 位的附加 ALU 单元提供支持，ALU 的使用受到指令集的控制，可以利用这种可编程性对并行性以及功耗进行优化。这些资源由 C55x CPU 中的地址数据流单元 (address data flow unit, AU) 和数据计算单元 (data computation unit, DU) 进行管理。

## 3. 指令集

TMS320C55x 采用的是一种低功耗、通用信号处理体系结构，指令集的设计目的包括执行效率高、易于使用和编码紧凑。C55x 体系结构支持变长指令，这样可以提高代码密度。指令缓冲单元 (instruction buffer unit, IU) 从内部或外部存储器读取 32 位的程序，并对指令进行排队，等待传送给程序单元 (program unit, PU)。程序单元对指令进行译码，将任务发送给 ALU 和 DU，并对完全保护的流水线进行管理。体系结构中还有一个可配置的 cache，用来减少对外部存储器的访问，从而提高数据吞吐量，并降低系统功耗。

高度并行化的体系结构是对 C55x 指令集的进一步补充，可以提高代码密度，同时又降低每次操作所需要的时钟周期数。高效、紧凑的指令集与高度并行化的体系结构相结合，实现了一种高性能的信号处理引擎，同时又能降低代码大小和功耗。

C55x 指令集还有一些灵活的特性，来提高易用性和程序效率：采用了强大的寻址模式，极大地减少了信号处理算法的指令数量，包括绝对寻址模式、寄存器间接寻址模式、直接寻址模式（也称为偏移量）；3 操作数指令格式，既支持内存地址，也支持寄存器地址，还有助于提高指令密度。

所有对数据进行移动操作的 C55x 指令都可以支持任何一种寻址模式和操作数格式。这种性质可以提高高级语言编译器的效率，并简化汇编语言编程工作。指令集中还有一些语法，支持程序员或编译器对多条指令进行调度，实现并行执行。这些指令集特点简化了程序员的任务，优化了 C55x 代码的有效性，缩短了产品的开发周期。

C55x 中有助于提高处理能力和增加代码密度的关键特性是它的高效实现方法。这种实现方法使用可变的指令长度，以此提高代码密度和总线使用效率。在处理器中设置了多个计算单元可以并行进行计算，以此减少每个操作所需要的执行周期数。双乘累加单元可以在一个周期内执行 2 次 17 位  $\times$  17 位的 MAC 操作，40 位的 ALU 可以用于处理 32 位的数据，也可以分开对 2 个 16 位的数据进行处理。

C55x 中还有另外一个 16 位的 ALU，用于执行通用算术运算，进一步提高了运算的并行性，并增加了灵活性。C55x 采用调整的 Harvard 体系结构，其中包含 1 条程序总线和 3 条独立的读数据总线，可以同时将操作数传输给不同的计算单元。这种高级并行性和高效的指令编码使整个处理器效率达到最优，而同时几乎不会降低性能。

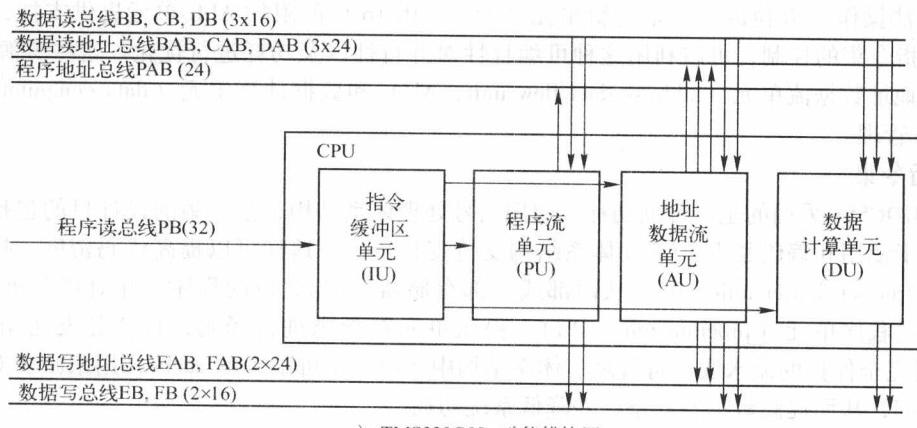
### 15.1.4 主要功能单元

C55x 体系结构围绕指令缓冲单元 (IU)、程序流单元 (PU)、地址数据流单元 (AU) 和数

据计算单元 (DU) 4 个主要模块进行构建。这些功能单元通过多条专用的内部总线，进行相互之间以及与内存中程序和数据的信息交换。

图 15-4a 和图 15-4b 给出了 C55x 中的主要功能模块和总线结构，程序通过 24 位的程序地址总线 (program address bus, PAB) 和 32 位的程序读总线 (program read bus, PB) 读取。

各个功能单元通过 3 条 16 位的数据读总线从内存中读取数据，分别称为 B 总线 (BB)、C 总线 (CB) 和 D 总线 (DB)。每一条数据读总线还有一条相关的 24 位数据读地址总线 (BAB、CAB 和 DAB)。单操作数读取在 D 总线上执行，双操作数读取使用 C 总线和 D 总线；B 总线是第三条读取通道，可以用于协助双重乘操作。



a) TMS320C55x 功能模块图

执行的操作	使用的总线
取指令	24位的程序地址总线 (PAB) 32位的程序读总线 (PB)
IU、AU、DU和PU从数据存储器中读	3条16位的数据读总线：BB、CB和DB
单操作数读	D总线
双操作数读	C总线和D总线
双重乘操作的协同读	B总线
程序和数据写	2条16位的数据写总线——EB和FB，以及相关的24位数据写地址总线——EAB和FAB

b) TMS320C55x 总线使用

图 15-4 (经德州仪器许可使用)

程序和数据的写操作通过 2 条 16 位的数据写总线执行，称为 E 总线 (EB) 和 F 总线 (FB)。每条写总线都有与之相关的 24 位数据写地址总线 (EAB 和 FAB)。在 C55x 中还有其他总线，专门为 DMA 控制器和其他外设控制器服务。

### 1. 指令缓冲单元

指令缓冲单元 (I 单元) 将程序代码接收到该单元内部的指令缓冲队列中，并对指令进行译码。接下来 I 单元会向其他单元传输相应的信息，进行进一步的处理，这些单元包括程序流单元、地址数据流单元和数据计算单元。CPU 从内存中取得 32 位的数据，放入指令缓冲队列 (instruction buffer queue, IBQ) 中。IBQ 最多可以容纳 64 字节的指令，等待译码；一次可以向指令译码单元传输 6 个字节，接下来译码器会向 CPU 中其他的主要功能单元发送控制信号。

TMS320C55x 的指令缓冲单元负责从内存中取指令流，并传送到 CPU 中。在每一个 CPU 周期中，I

单元从 32 位的程序总线上获得 4 个字节的程序代码，并对缓冲队列中 1~6 个字节的代码进行译码。

接下来，I 单元将译码后的信息传送给 P 单元、A 单元和 D 单元，对指令进行进一步的处理。图 15-5 给出了 I 单元的框图。

在流水线的指令预取段，CPU 从程序存储器中取得 32 位的代码，放入指令缓冲队列中。当 CPU 已经准备好对指令进行译码时，会将最多 6 个字节的指令从队列传送给指令译码器。指令缓冲区在同一时刻最多可以容纳 64 个字节的代码，可以保持程序流的连续性，进而提高 CPU 的性能。

指令缓冲队列还协助重复执行指令，重复或循环执行的指令保存在队列中的一个代码块。这种循环方式无论是在性能上，还是在功耗上，效率都是非常高的，因为一旦代码被装载到队列中，执行循环的过程中就不再需要进行存储器读操作。

指令缓冲队列的另外一个优点是，当程序流控制指令在进行条件测试（条件调用、条件返回或条件跳转）时，它可以根据时机进行取指操作。这种功能可以尽量减小由于程序流的不连续造成的开销，因为没有必要清空流水线。那些本来可能要用于清空流水线而损失掉的时钟周期，现在可以用于处理有效的指令。

在流水线的译码段，指令译码器从指令缓冲队列中获得最多 6 个字节的程序代码，然后对这些代码进行译码。指令译码按照它们在指令缓冲队列中的接收顺序进行，I 单元并不执行动态调度。这种方式使得执行时间可预测，而这一点对于实时嵌入式系统的设计来说是非常重要的。

C55x 指令集的编码长度可变，指令长度可以为 1~6 个字节。所有指令并不是用相同的位宽进行编码，简单的指令字短一些，而复杂的指令字长一些。指令译码器用于识别指令的边界，可以对 8 位、16 位、24 位、32 位、40 位和 48 位的指令进行译码。这种译码方式可以极大地提高代码密度，减少程序存储器的使用。

### 指令 cache

图 15-6 给出了 DSP 系统中的 I-cache 结构。CPU 状态寄存器 ST3\_55 中有 3 个 cache 控制位，分别用于 I-cache 的使能、冻结和清空。CPU 可以通过 I-cache 中的一组寄存器对 I-cache 进行配置，并检查它的状态。I-cache 中有一个两路 cache，用于保存指令。两路 cache 采用两路组相联的映射方式，最多可以保存 16K 字节 512 组，每一组有两行，每行有 32 位的字。在两路 cache 中，每一行都有唯一的标示用于分辨。

当从外部存储器取指令时，CPU 通过程序读数据总线（P 总线）向指令 cache（I-cache）发送一个 32 位的访问请求。如果指令 cache 被禁止，访问请求将直接到达 IPORT，然后发送到外部存储器接口（external memory interface，EMIF）；EMIF 必须从外部存储器中读取 32 位的指令，然后传送给 IPORT，最后再将数据发送给 CPU。

当指令 cache 启用时，可能有两种情况发生。如果发生了 cache 命中，指令 cache 将立即处理 CPU 的请求，不用从外部存储器中读取数据。如果发生了 cache 失效，指令 cache 将通过 IPORT 从 EMIF 请求 4 个 32 位的指令。EMIF 将从外部存储器中读取 4 个 32 位的字，并将数据通过 IPORT 传送给指令 cache。然后指令 cache 将读取的数据发送给 CPU，并更新 cache 的内容。

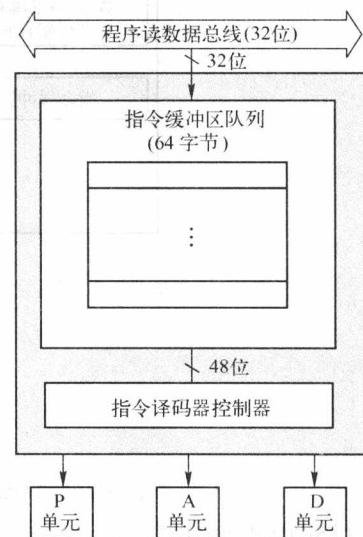


图 15-5 TMS320C55x 指令 I 单元框图  
(经德州仪器许可使用)

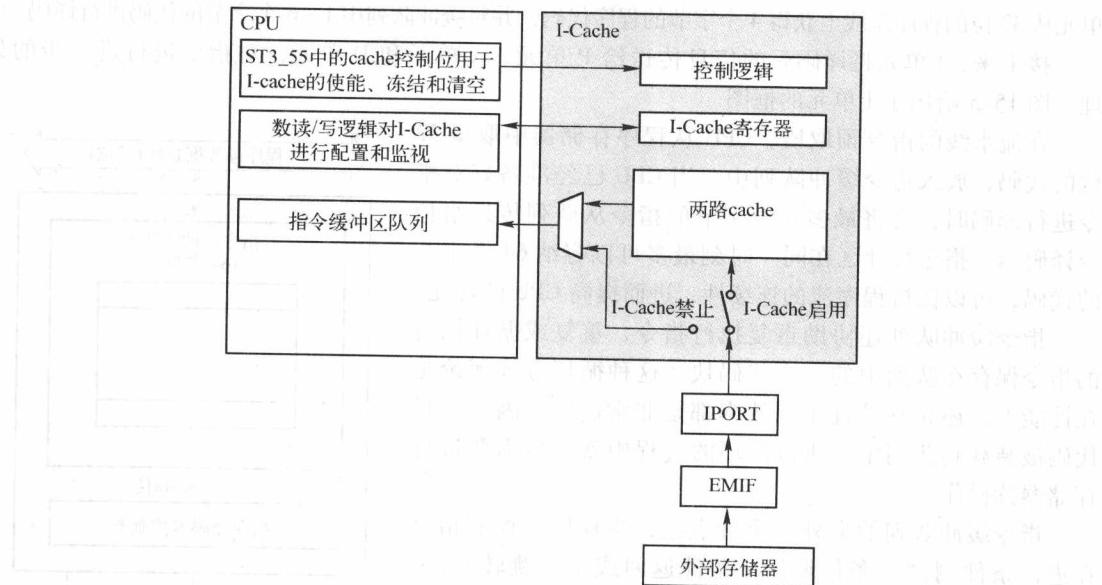


图 15-6 TMS320C55x 指令 cache (经德州仪器许可使用)

C55x 中灵活的指令 cache 还有一种可配置 cache 的功能，可以针对不同类型的代码优化 cache 的操作。提高 cache 命中率，意味着可以减少对外部存储器的访问，降低整个系统的功耗。指令 cache 的 burst-fill 功能可以尽量减少外部存储器的访问，进而避免由于访存所带来的性能和功耗损失。

## 2. 程序流单元

程序流单元从 I 单元中获得指令，并协调程序流操作，具体包括：

- 解释条件指令的条件；
- 确定分支（跳转）地址；
- 当发生中断请求时，初始化中断服务；
- 管理单重复和块重复操作；
- 管理并行指令的执行。

C55x 程序流单元（或 P 单元）控制着程序中指令序列的执行，它产生程序存储器的取指地址，并对一些操作进行控制，例如硬件循环、分支和条件执行。这个单元中还设计了管理指令流水线的逻辑，还有 4 个状态寄存器，用于控制和监视 CPU 的多个功能。P 单元组成部分的功能可以提高 C55x 的周期效率。图 15-7 给出了 P 单元的框图。

在 P 单元中，程序地址生成逻辑产生 24 位的地址，用于从程序存储器中读取指令；代码在内存中的放置没有对准的限制，因为 P 单元支持字节寻址；24 位的地址可以寻址 16M 字节的 C55x 程序范围，足以容纳大的程序。

P 单元通常使用程序计数器产生连续的地址，用来跟踪程序中指令的执行。然而，该逻辑还会产生不连续的地址，用来支持程序控制操作，例如：

- 分支；
- 调用；
- 返回；
- 硬件循环（重复）；
- 条件执行；
- 中断服务。

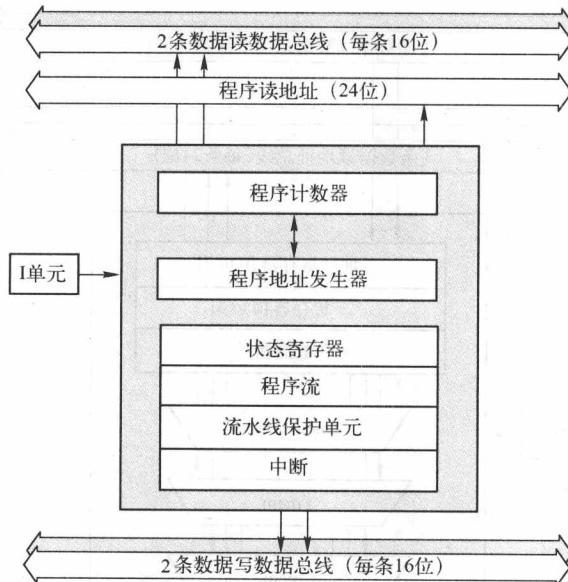


图 15-7 TMS320C55x 程序流单元框图（经德州仪器许可使用）

P 单元可以在尽量不影响流水线性能的前提下高效执行程序流。P 单元的地址生成逻辑与 CPU 中的其他单元是完全独立的。由于这个原因，在流水线的早期就可以计算出分支的目标地址，也可以测试条件分支的条件，以尽量减少分支延迟。

这种并行性使得程序控制指令可以在与数据处理指令一样的流水线段中执行，使得 C55x 的性能比先前的体系结构有了很大的提高，因为在先前的体系结构中，只能通过加入延迟槽的方式来提高分支的性能。P 单元中还有其他提高程序控制性能的特性，包括预测分支逻辑，还包括另外一个程序计数器，专门用于从子程序或中断服务子程序中快速返回。

P 单元的循环功能包括单条指令或者指令块的重复执行。通过将一个块重复操作与另外一个块重复操作嵌套执行，在 C55x 中可以有 3 个层次的硬件循环，一个或两个重复块中包含一次重复。P 单元有支持条件重复的硬件结构。

P 单元有一个专门的逻辑用于流水线保护。除了对控制相关进行处理之外，P 单元还为防止写后读 (write-after-read, WAR) 和读后写 (read-after-write, RAW) 数据相关提供了完全的保护。当 C55x 中的指令流发生这样的数据相关时，流水线保护逻辑将插入几个周期，保证操作的正确顺序和程序的正确执行。

### 3. 地址数据流单元

地址数据流单元为数据空间的读和写操作产生地址。该单元中设置了必要的逻辑和寄存器，用于为 3 条数据读地址总线和两条数据写地址总线产生地址。单元中还有一个具有移位功能的通用 16 位算术逻辑单元 (ALU)，图 15-8 给出了地址流 (A) 单元的框图。

单元中还有 8 个辅助寄存器，用作地址指针和系数数据指针寄存器，为系数表提供专门的指针。循环寻址控制的寄存器也是由 A 单元管理的。A 单元中还有一个 16 位的 ALU，能够执行算术、逻辑、移位和饱和操作。

16 位的 ALU 单元可以将简单的算术操作与 D 单元中的复杂操作并行执行。它接收来自 I 单元的立即数，并与存储器、A 单元寄存器、D 单元寄存器和 P 单元寄存器进行双向通信。在 A 单元中，ALU 可以控制 4 个通用 16 位寄存器，或者任何一个地址生成寄存器。4 个通用寄存器可以提高编译器的效率，并减少对存储器的访问。

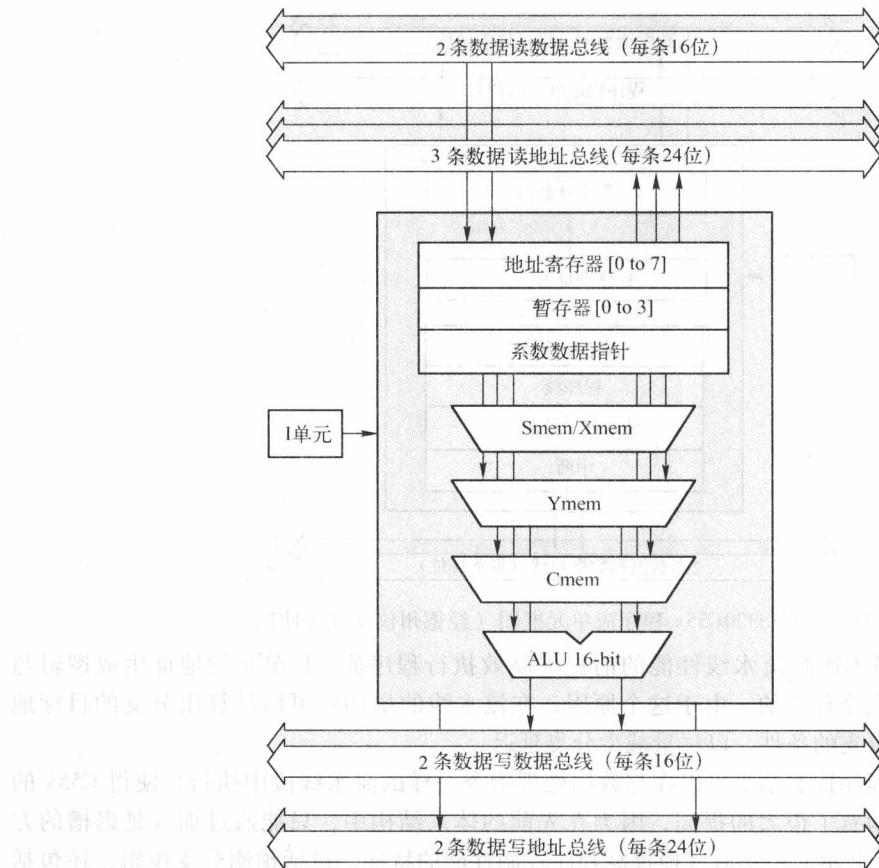


图 15-8 TMS320C55x 地址流单元框图（经德州仪器许可使用）

通用 ALU，或者 3 个寻址寄存器 ALU (ARAU) 中的任何一个，可以修改直接寻址的 9 个寻址寄存器。ARAU 为 C55x 的 3 条数据读总线中的每一条都提供单独的地址发生器。这种并行性可以在一个 CPU 周期内将两个 16 位的操作数和 1 个 16 位的系数读入 D 单元中。A 单元中还有专用寄存器，支持采用寻址方式的指令进行循环寻址；最多可以同时使用 5 个独立的循环缓冲区，最多 3 个独立的缓冲区长度。对这些循环缓冲区也没有地址对准的限制。

#### 4. 数据计算单元

数据计算 (D) 单元是 CPU 中的一个重要单元，用于数据处理。3 条数据读总线可以给两个乘累加 (MAC) 单元和 40 位的 ALU 提供数据，中间结果可以存储在 4 个 40 位累加器寄存器的任何一个中。该单元实现了并行性，尽量减少每个任务需要的执行周期，可以高效地执行信号处理算法。图 15-9 给出了 D 单元的框图。

C55x 的计算能力关键在于双 MAC 体系结构。一个 MAC 单元中有一个乘法器和一个具有饱和逻辑的专用加法器。在一个周期中，每一个 MAC 单元可以执行一个 17 位  $\times$  17 位的乘法和一个 40 位的加法或减法操作，可以选择 32/40 位的饱和。三条数据读总线可以给两个 MAC 单元分别提供数据流和一个共用的系数流，MAC 单元的计算结果可以放置在 D 单元中 4 个 40 位的累加器的任何一个中。对于执行数据块滤波算法和其他信号处理的应用，双 MAC 数据计算单元 (D 单元) 功能可以极大地提高 C55x 的性能。

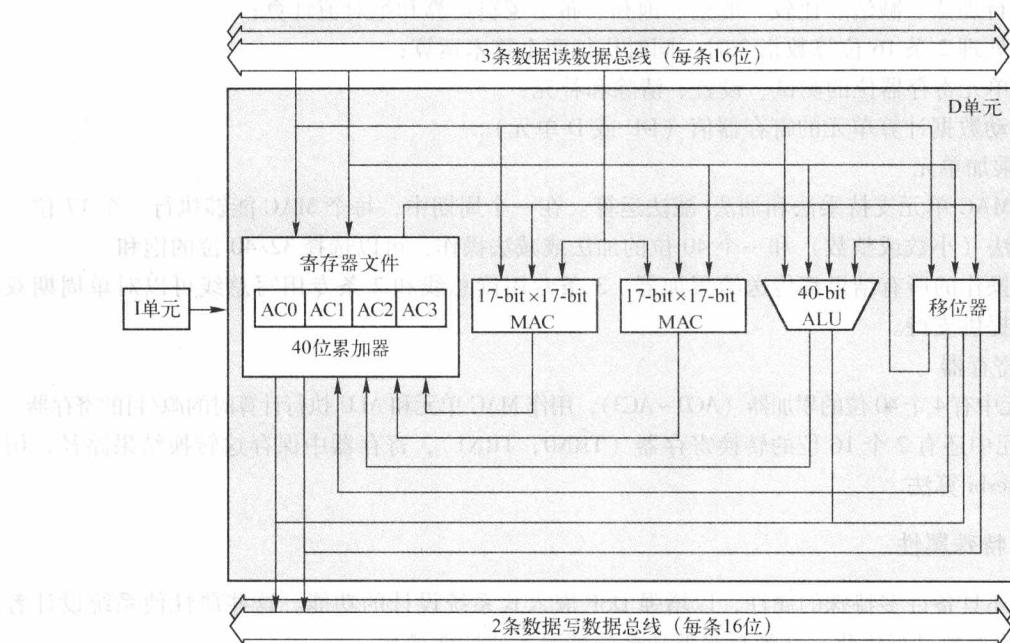


图 15-9 TMS320C55x 数据计算单元框图（经德州仪器许可使用）

D 单元中还有一个 40 位的 ALU，与 MAC 单元是完全分开的。D 单元的 ALU 可以对累加器中的 40 位数据执行算术或逻辑运算，或者用来同时执行两个 16 位的算术运算。

除了接收来自 D 单元的 40 位累加器寄存器中的数据之外，ALU 还可以接收来自 I 单元的立即值，并与存储器、A 单元寄存器或 P 单元寄存器进行双向通信。

筒形移位器进一步增强了 D 单元中 MAC 和 ALU 单元的功能。这个移位器可以将 40 位的累加器值最多向左移动 31 位，或向右移动 32 位。它可以接收来自 I 单元的立即值，并与存储器、A 单元寄存器或 P 单元寄存器进行双向通信。此外，它还将移位后的值传输给 D 单元的 ALU，进行进一步处理。

D 单元计算的结果可以通过 2 条 16 位的数据写总线写入存储器中。这些总线与 A 单元地址发生逻辑一起，能够在一个 CPU 周期内执行 2 个 16 位的写操作存储器操作或者 1 个 32 位的写存储器操作。这样的数据吞吐量足以支持 D 单元的实时处理功能。

### (1) 移位器

D 单元的移位器可以进行如下操作：

- 将 40 位的累加器值最多向左移动 31 位，或向右移动 32 位，移位的次数可以从暂存器中读取，或者可以在指令中当做常数来使用；
- 将 16 位的寄存器、存储器或者 I/O 空间值最多向左移动 31 位，或向右移动 32 位；
- 将 16 位的立即数最多向左移动 15 位，移动的次数在指令中固定给出；
- 规格化类机器值；
- 提取和扩展位域，并执行位运算；
- 循环移位寄存器值；
- 在将它们存储到数据寄存器之前，对累加器值取整和/或饱和。

### (2) 算术逻辑单元

D 单元中有一个 40 位的 ALU，可以接收 I 单元的数据，并将计算结果传送给其他功能单元。D 单元 ALU 的功能有：

- 执行加法、减法、比较、取整、饱和、布尔逻辑运算和绝对值计算；
- 当处理2条16位算数指令时，同时执行两个算术运算；
- D单元寄存器位的测试、设置、清除和补充；
- 移动数据计算单元的寄存器值（DU或D单元）。

### (3) 乘加单元

两个MAC单元支持乘法和加法/减法运算。在一个周期中，每个MAC能够执行一个17位×17位的乘法（小数或整数）和一个40位的加法或减法操作，可以选择32/40位的饱和。

MAC操作的所有结果都传送给累加器。3条专用读总线和2条专用写总线可以对单周期双MAC操作提供支持。

### (4) 寄存器

D单元中有4个40位的累加器（AC0~AC3），用作MAC单元和ALU执行计算时的源/目的寄存器。

D单元中还有2个16位的转换寄存器（TRN0, TRN1），寄存器中保存这转换结果路径，用来执行Viterbi算法。

## 15.1.5 特殊属性

C55x还具备许多特殊的属性，以增强DSP嵌入式系统设计的功能。这些属性使系统设计者可以对系统参数进行优化，在消耗最低功耗的情况下使性能最高。

### 低功耗

在处理器中进行了一系列的处理、设计和体系结构方面的改进，降低了整体功耗。这些针对C55x的设计改进不仅获得超低功耗，还极大地提高了性能。这样对于需要使用DSP的移动设备，可以大幅度增强系统的功能性。当工作在最高频率下时，功耗只有1.1V，可以显著延长电池寿命，例如医学设备。

## 15.1.6 低功耗设计

40位的ALU与C54x的体系结构相容，用于执行主要的计算任务。C55x体系结构中还有另外一个的16位ALU，可以用于处理小型的算术和逻辑运算。由于指令集很灵活，16位的ALU可以执行简单的计算或逻辑/位运算，这样可以降低功耗；并且还可以减小每个任务的执行周期，因为两个ALU是并行运行的。

### 1. 存储器访问

无论是内部存储器访问，还是外部存储器访问，都会造成大量的功耗损失。尽量减少完成一项任务所需用的访存次数，可以进一步降低功耗。C55x减少了向CPU读取指令的必要访存次数。在C55x中，取指操作可以作为32位的访存操作来处理。

此外，变长指令意味着每取来一个32位数据，可能会获得不止一条指令。变长指令增加了代码密度，并因为根据需要的信息设置指令大小而节约了功耗。这种指令集设计方法和体系结构相结合，在保持应用程序最高性能的前提下，尽量降低功耗。

### 2. 自动电源管理机制

C55x内核处理器可以对片上外设和存储阵列的功耗进行有效管理。这种电源管理是完全自动的，并且对用户透明。这种管理对软件或应用程序的计算性能没有任何影响，又是一种在不影响性能的情况下降低功耗的方法。

当片上存储器阵列未被访问时，就会将这些存储器切换到低功耗模式。当接收到访问请求时，存储器会立即返回正常操作模式，并完成存储器访问操作。如果对该存储器没有进一步的访问请求，存储器将返回低功耗状态，直到再次需要访问为止。

### 3. 低功耗改进

处理器提供类似的方法控制片上外设。当外设处于非活跃状态，并且 CPU 不需要它们时，可以进入低功耗状态。外设也可以无延迟地对处理器请求做出响应，退出低功耗状态。这种功耗管理是对软件控制低功耗状态的补充，软件控制低功耗状态是由外设的 IDLE 域提供的。

节约功耗的关键是，在应用处于空闲或低活动的状态时，降低系统的功耗。通过实现用户可控制的 IDLE 域，C55x 改进了低活动功耗管理。这些域是设备的组成部分，在软件的控制下，可以有选择地启用或禁止。当某个域被禁止后，将进入低功耗 IDLE 状态，在这个状态下，寄存器或内存中的内容将继续保持。当域被启用后，将会返回普通操作模式，每一个域都可以单独启用和禁止，使应用程序可以尽量高效地对低活动电源情况进行管理。

在最初的 C55x 处理器中，配置为独立 IDLE 域的设备域有 CPU、DMA、外设、外部存储器接口（EMIF）、指令 cache 和时钟发生电路。

### 4. 节约电能

如下 3 种特性有助于节约功耗：

- 软件可编程的空闲域，提供了可配置的低功耗模式；
- 自动电源管理；
- 先进的低功耗 CMOS 工艺。

### 5. 空闲域

C55x 的体系结构很灵活，通过软件可编程域可以动态地节约电能。处理器中的电路模块被组织为空闲域。每一个域都可以正常运行，也可以被设置为低功耗空闲域。当执行下一条 IDLE 指令时，由空闲控制寄存器（idle control register, ICR）确定将哪个域设定为空闲状态。六个域如下：

- CPU 域；
- DAM 域；
- 外设域；
- 时钟发生器域；
- 指令 cache 域；
- EMIF 域。

当某个域的功能暂时不需要时，就可以将其设置为低功耗状态，这样用户就可以根据实际的操作自动调整功耗。要注意的是，当某个域处于空闲状态时，该域将不再起作用。但是外设域是一个例外。在外设域中，每一个外设都有一个空闲使能位，控制着某个外设在空闲状态时是否对外界变化做出响应。

因此，当外设域处于空闲状态时，各个外设可以单独配置为空闲，或者仍然保持活跃。修改 ICR（如果 CPU 和生成域没有处于空闲状态）或者发生外部中断，都可以导致退出空闲域。

### 6. 高级技术

除了体系结构和指令集的改进可以降低功耗之外，C55x 处理器还通过低功耗 CMOS 技术进一步降低功耗。CMOS 处理器基于低功耗 CMOS 技术进行设计，工作电压只有 1.5 V 和 0.9 V，但这些低电压处理器仍然可以与标准的 3.3V CMOS 芯片直接进行接口。

#### 15.1.7 处理器片上外设

图 15-10 列出了定点数字信号处理器（DSP）TMS320C55x 的片上处理器外设，根据应用目标的不同，可以选择不同的外设组合方式。对于一个给定的处理器，一些外设可以共享引脚，互斥使用这些外设。

外设
模数转换器 (ADC)
具有PLL的时钟发生器
直接存储器存取 (DMA) 控制器
外部存储器接口 (EMIF)
主机接口 (HPI)
指令cache
内置集成电路 (I2C) 模块
多通道缓冲的串行接口 ( McBSP)
多媒体卡/SD卡控制器
电源管理/空闲配置
实时时钟 (RTC)
通用定时器
watchdog 定时器
通用异步收发器 (UART)
通用串行总线 (USB) 模块

图 15-10 TMS320C55x 的外设 (经德州仪器许可使用)

## 1. 片上存储器

片上存储器由 3 部分组成：双口 RAM (DARAM)，在一个周期中支持两次存储器访问，DARAM 的最大容量为 128K 字 (256 字节)；单端口 RAM (SARAM) 在一个周期中只能进行一次存储访问，容量为 32K 字 (64K 字节)；ROM 模块，用于程序指令或数据的非易失性存储。

## 2. 模数转换器

ADC (见图 15-11) 将模拟输入信号转换为数字值，提供给 DSP 使用。ADC 在同一时刻可以对 4 个输入 (AIN0 ~ AIN3) 中的一个进行采样，并将采样值转换为一个 10 位的数字表示 (ADCDATA)。ADC 的最高采样频率为 21.5 kHz，因此 ADC 可以采样变化频率比较低的模拟信号。

ADC 可以用于采样用户接口板上两端的电压，或用于监视电池的电压下降，并不将采样数据用于信号处理。要注意的是，ADC 只有在 TMS320VC5509A 中才有。

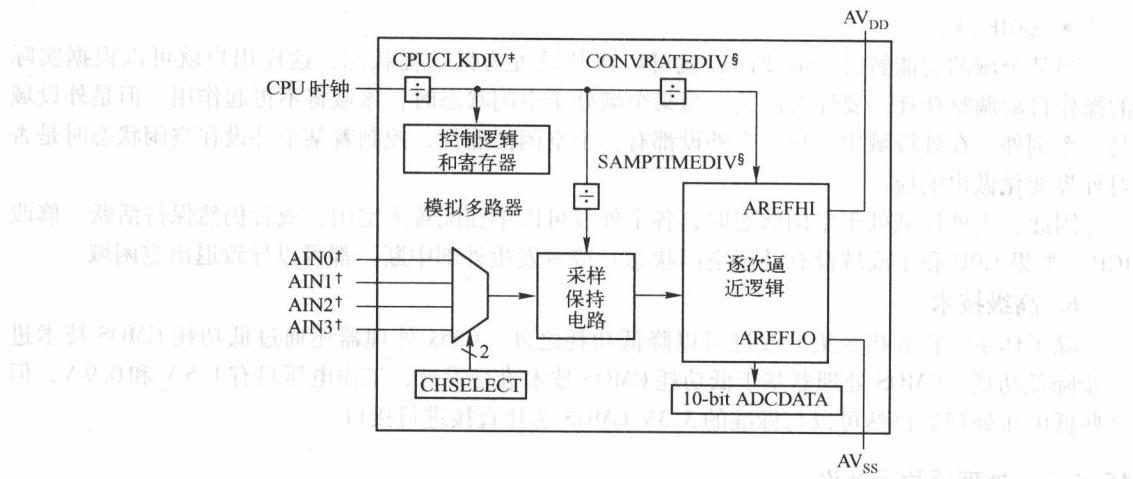


图 15-11 TMS320C55x 模数转换器 (经德州仪器许可使用)

## 3. DSP 时钟发生器

DSP 时钟发生器为 DSP 提供时钟信号，该时钟信号基于 CLKIN 引脚的输入时钟信号产生。

在时钟发生器中有一个数字锁相环 (PLL)，可以被启用或禁止。对时钟发生器进行配置，可以产生特定频率的 CPU 时钟信号（见图 15-12）。

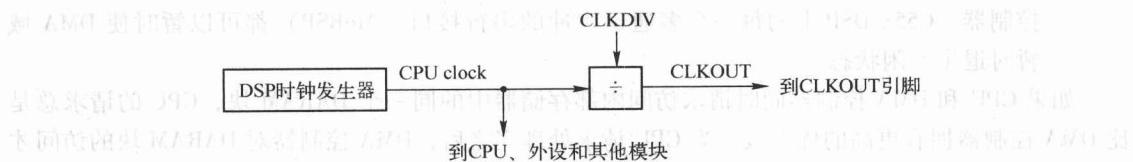


图 15-12 TMS320C55x 时钟发生器（经德州仪器许可使用）

时钟发生器中有一个时钟模式寄存器 CLKMD，用于控制和监视时钟发生器的运行。例如，可以设置 CLKMD 中的 PLL ENABLE 位，实现在两个主要操作模式之间进行选择：

在旁路模式中，PLL 被旁路，输出时钟信号的频率等于输入时钟信号的频率除以 1、2 或 4。因为 PLL 被禁止，这种模式可以节约功耗。

在时钟模式中，输入频率可以被倍频或降频，产生需要的输出频率，输出时钟信号被锁相为输入时钟信号。如果时钟模式寄存器中的 PLL ENABLE 位被设置为 1，并且完成了锁相序列，则进入时钟模式。在锁相序列过程中，时钟发生器保持旁路状态。

#### 4. DMA 控制器

DMA 控制器与 CPU 并行运行，在同一个周期之内，可以在内部存储器、外部存储器和片上的外设之间传输 32 位的数据。从本质上来说，它的工作类似于 CPU 的协处理器。图 15-13 给出了 DMA 的基本框图。

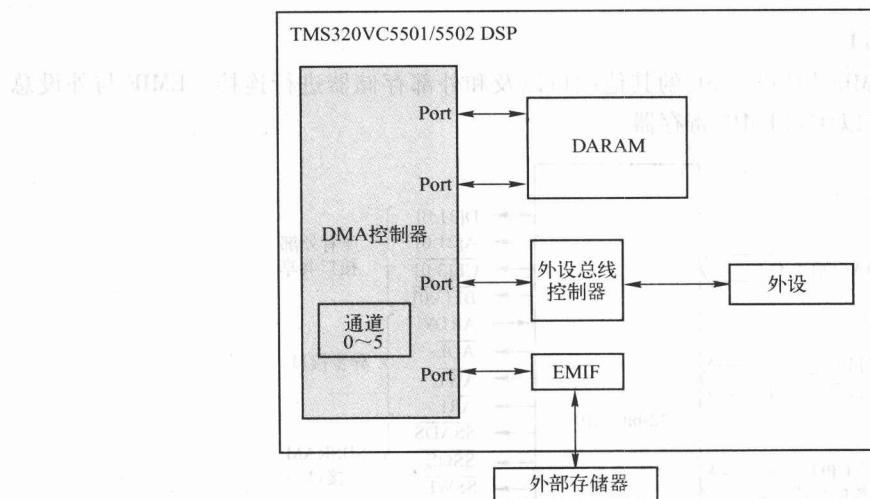


图 15-13 TMS320C55x DMA 控制器（经德州仪器许可使用）

DAM 控制器的主要特点如下。

- 与 CPU 相互独立运行。
- 4 个标准端口：两个用于内部双端口 RAM (DARAM)，一个用于外部存储器，一个用于外设。
- 6 个通道，DMA 可以通过这 6 个通道，跟踪 6 个独立模块在标准端口之间的数据传输。
- 给每个端口设置高优先级或低优先级。
- 事件同步。每个通道都可以根据所选择事件的发生进行 DMA 传输。
- 每个通道都有一个中断。当完成操作后，通道可以向 CPU 发送一个中断。

- 软件可选择的选项，用来更新数据传输的源地址和目的地址。
- 专用的空闲域。可以将这个域关闭，将 DMA 控制器设置为低功耗状态。如果需要 DMA 控制器，C55x DSP 上的每一个多通道缓冲的串行接口（ McBSP）都可以暂时使 DMA 域暂时退出空闲状态。

如果 CPU 和 DMA 控制器同时请求访问内部存储器中的同一个 DARAM 块，CPU 的请求总是比 DMA 控制器拥有更高的优先级。当 CPU 请求处理完之后，DMA 控制器对 DARAM 块的访问才会得到响应。关于每一个 DARAM 块的起始和结束地址，可以查阅芯片指南。

DMA 控制器有 6 条路径，称为通道，可以在 4 个端口之间传输数据（两个用于 DARAM，一个用于外部存储器，一个用于外设）。每一个通道从一个端口（源端口）读取数据，并将数据写到同一个或者另外一个端口（目的端口）。每一个通道都有一个先进先出（FIFO）缓冲区，将数据的传输过程分为如下 2 个阶段（见图 15-14）：

- 端口读访问。将数据从源端口传输到通道 FIFO 缓冲区。
- 端口写访问。将数据从 FIFO 缓冲区传输到目的端口。

另外，每个通道的 FIFO 缓冲区深度都是 8 个 32 位的字。



图 15-14 DMA 传输的两个阶段（经德州仪器许可使用）

## 5. 外部存储器接口

图 15-15 解释了 EMIF 如何与 DSP 的其他端口以及和外部存储器进行连接。EMIF 与外设总线控制器相连，CPU 可以访问 EMIF 寄存器。

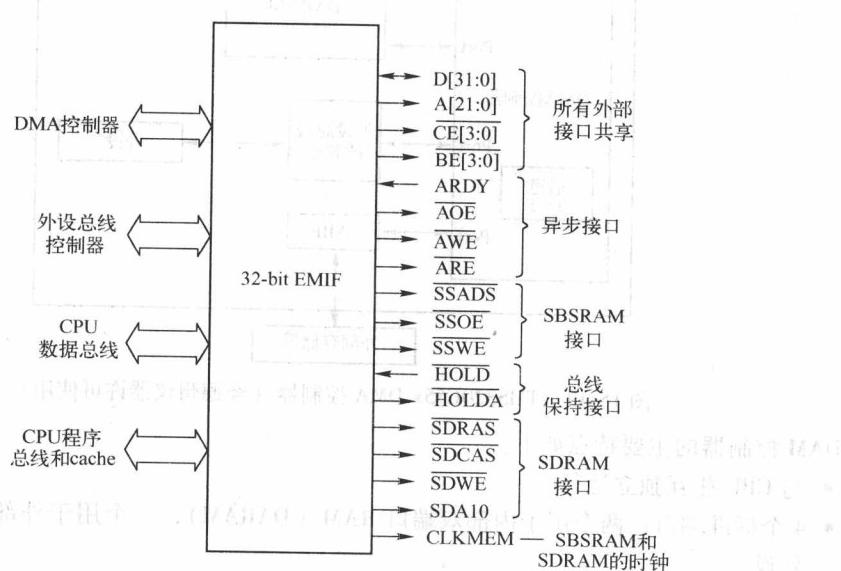
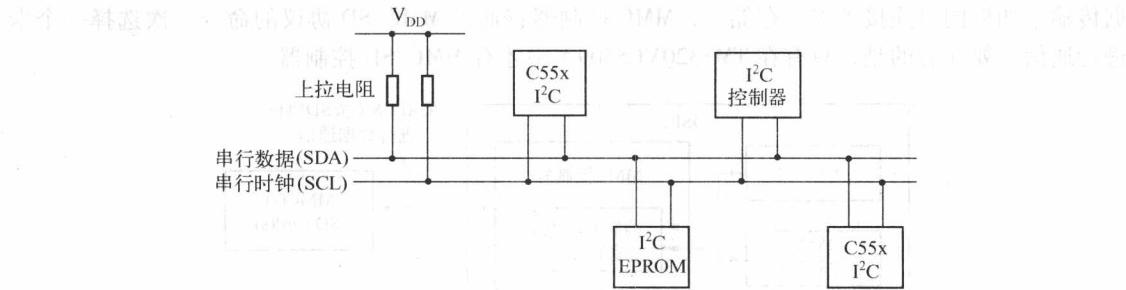


图 15-15 TMS320C55x 外部存储器接口（经德州仪器许可使用）

## 6. I<sup>2</sup>C 模块

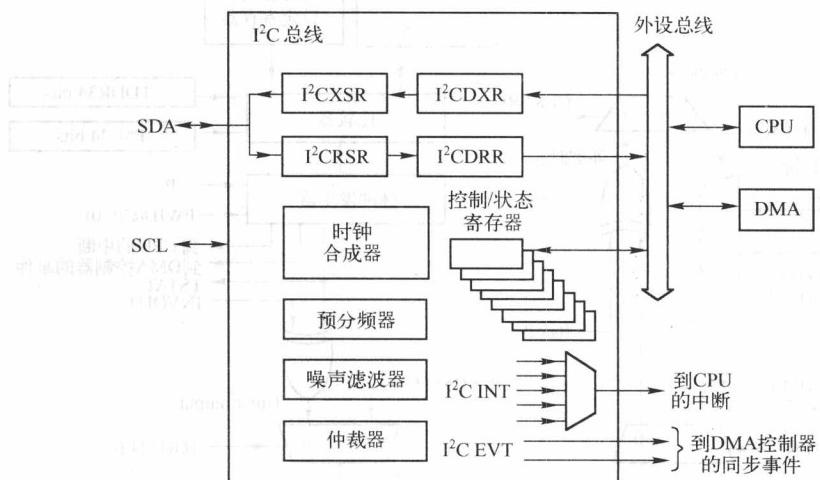
I<sup>2</sup>C 模块支持任何与 I<sup>2</sup>C 兼容的主设备或从设备。图 15-16 给出了多个 I<sup>2</sup>C 模块相连的例子，作为两个设备之间的两路数据传输。

图 15-16 I<sup>2</sup>C 设备互连（经德州仪器许可使用）

每一个设备，包括所有的 C55x DSP，都要通过 I<sup>2</sup>C 模块连接到 I<sup>2</sup>C 总线上，这些设备都由唯一的地址进行识别。根据设备的功能，每一个设备都可以作为发送设备或接收设备。当传输数据时，与 I<sup>2</sup>C 总线相连接的设备也可以看做主设备或者从设备。

主设备是发起总线上的数据传输过程的设备，并产生时钟信号来支持传输。在数据传输的过程中，由该主设备寻址的任何设备都作为从设备。I<sup>2</sup>C 模块支持多主模式，在这种模式下，一个或多个控制 I<sup>2</sup>C 总线的设备都可以连接到 I<sup>2</sup>C 总线上。

I<sup>2</sup>C 模块有一个串行数据引脚 (SDA) 和一个串行时钟引脚 (SCL)，用于数据通信，如图 15-17 所示。这两个引脚在 C55x 设备和连接到 I<sup>2</sup>C 模块的其他设备之间传递信息。SDA 和 SCL 引脚都是双向的，并且都要通过上拉电阻连接到高电平。当总线空闲时，两个引脚都为高。两个引脚的驱动器具有开漏配置，以实现线与功能。

图 15-17 TMS320C55x I<sup>2</sup>C 模块框图（经德州仪器许可使用）

## 7. 多媒体/SD 卡控制器

如图 15-18 所示，MMC 控制器在 CPU 或 DMA 控制器以及一个或多个存储器卡之间传送数据。CPU 或 DMA 控制器可以读或写 MMC 控制器中的控制寄存器和状态寄存器。根据需要，CPU 和/或 DMA 控制器可以将数据写入到 DSP 存储器或其他外设的寄存器中，也可以取回。CPU 可以读取状态寄存器的值监视数据行为，并对中断请求作出响应。

DMA 控制器可以通过两个 DMA 事件得知数据接收/发送的状态。使用一条双向数据线（根据 MMC 协议）或者 4 条并行数据线（根据 SD 协议），可以在 MMC 控制器和存储卡之间进行数

据传输。如果同时连接了多个存储卡，MMC 控制器将通过 MMC/SD 协议的命令一次选择一个卡进行通信。要注意的是，只有在 TMS320VC5509A 中才有 MMC/SD 控制器。

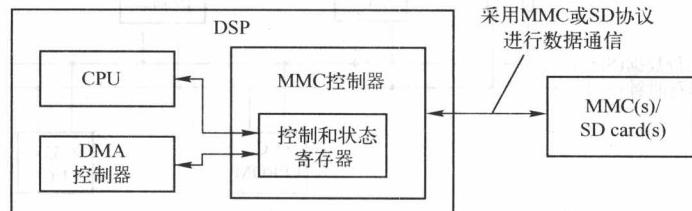


图 15-18 TMS320C55x MMC 控制器（经德州仪器许可使用）

### 8. 可编程定时器

DSP 中有三个可编程的定时器，如图 15-19 所示。两个通用（general-purpose, GP）定时器，定时器 0 (TIM0) 和定时器 1 (TIM1)。每一个 GP 定时器都可以通过全局定时器控制寄存器 1 (GCTL1) 中的定时器模式位，配置为 64 位定时器、双 32 位级联定时器或者双 32 位非级联定时器三种模式之一。

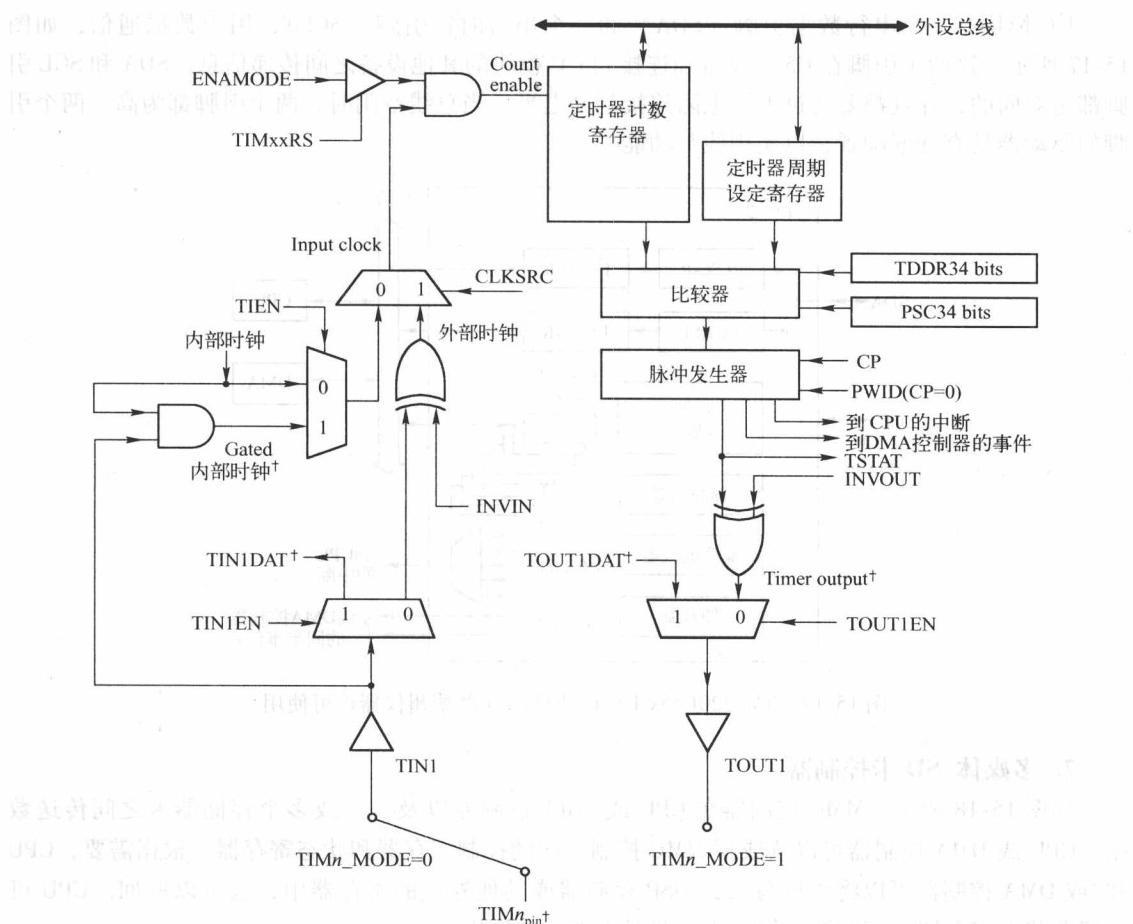


图 15-19 TMS320C55x 生成内部定时器时钟（经德州仪器许可使用）

两个 GP 定时器都不支持 watchdog 定时器模式。复位之后，两个 GP 定时器被配置为 64 位的定时器，第三个定时器可以配置为 GP 定时器，也可以配置为 watchdog 定时器。

复位之后，第三个定时器被配置为 64 位的 GP 定时器。每一个定时器都可以用定时器引脚上的外部时钟驱动，也可以用内部时钟驱动。当选择内部时钟驱动时，定时器时钟产生框图如图 15-18 所示。根据器件说明书，DSP 时钟发生器的时钟源来自外部。由 DSP 时钟发生器产生的时钟是快速外设时钟 (SYSCLK1)。根据器件说明书，SYSCLK1 是 CPU 时钟的降频。定时器内部有一个时钟分频器，根据预先设定的分频系数对 SYSCLK1 进行分频，产生定时器时钟。

## 9. UART

UART 外设（见图 15-20）是基于工业标准 TL16C55x0 的异步通信部分，该标准对 TL16C450 标准进行了功能升级。上电以后，UART 的功能类似于 TL16C450（单字符或者 TL16C450 模式），

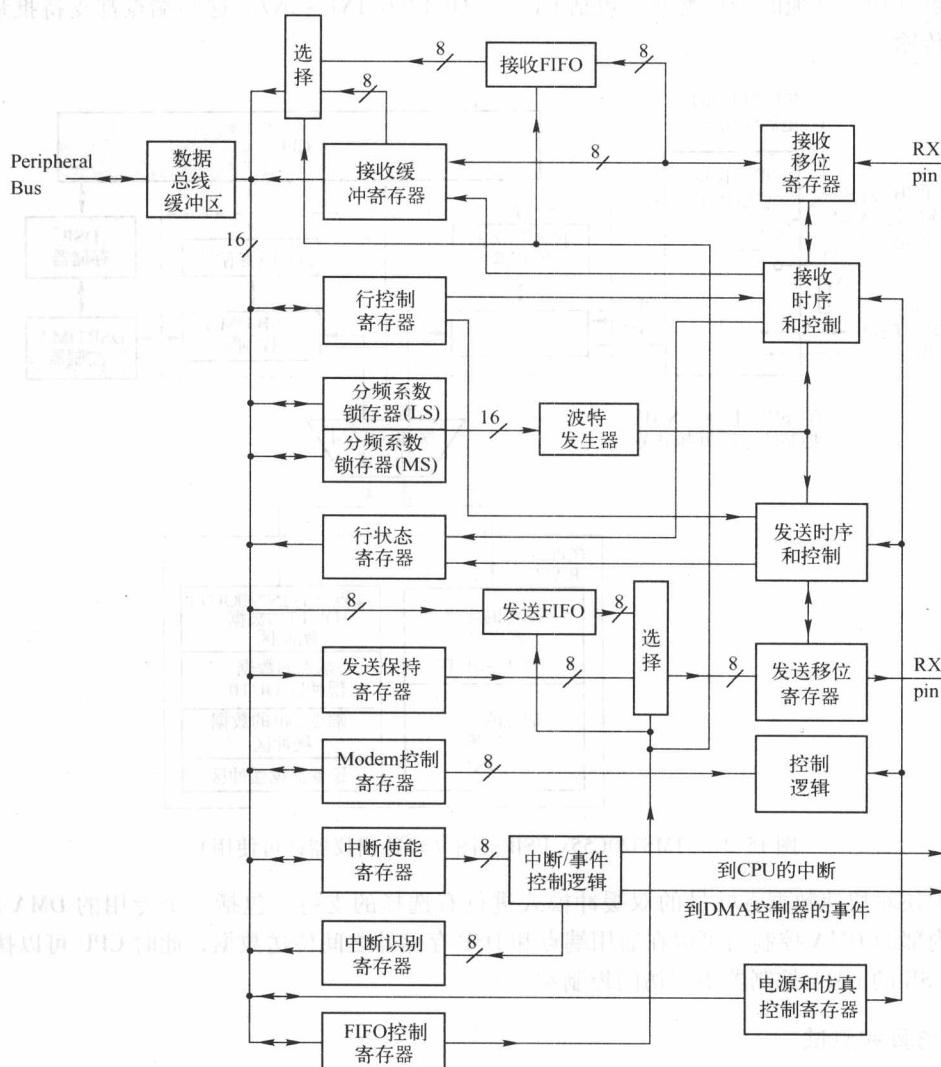


图 15-20 TMS320C55x UART 功能图（经德州仪器许可使用）

可以被设置为 FIFO (TL16C55x0) 模式。这样就没有必要设计专门的软件对接收或发生的数据进行缓冲，减轻了 CPU 的负担。接收和发送 FIFO 最多可以存储 16 个字节，接收器 FIFO 中的每一个字节包括 3 个位表示出错状态。

UART 将接收的数据进行串并转换，并将来自 CPU 的数据进行并行/串行转换，CPU 可以随时读取 UART 的状态。UART 还具有控制功能以及一个处理器中断系统，通过配置，可以尽量减少软件对通信链接的管理。UART 有一个可编程的波特发生器，可以对 UART 输入时钟进行分频，分频系数可以在 1~65535 之间，为内部发送接收逻辑生成一个 16 字节的参考时钟。

## 10. USB 模块

USB 模块与 USB 2.0 兼容，在 TMS320C5509A 中设置的是全速 (12 Mbit/s) 从模式（见图 15-21）。USB 模块有 16 个端点，两个控制端点（只用于控制传输）是 OUT0 和 IN0，另外 14 个通用端点（用于其他的传输类型）包括 OUT1~OUT7 和 IN1~IN7。这些端点都支持批量、中断和同步传输。

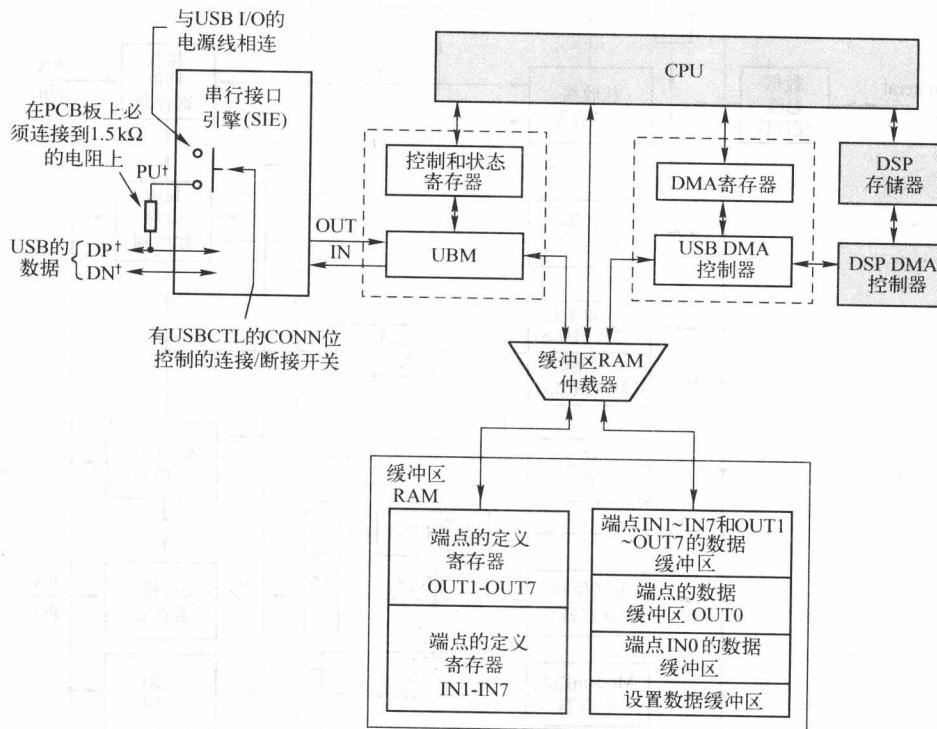


图 15-21 TMS320C55x USB 框图（经德州仪器许可使用）

USB 中还对快速数据吞吐量的双缓冲模式进行有选择的支持，包括一个专用的 DMA 通道。USB 模块内部的 DMA 控制器可以在通用端点和 DSP 存储器之间传送数据，此时 CPU 可以执行其他任务（USB 的 DMA 控制器不能访问控制端点）。

### 15.1.8 仿真和测试

芯片还有基于扫描的仿真功能，支持程序执行历史、追踪最近的程序计数器值和停顿（追踪 FIFO）功能，这样可以通过集成的开发环境进行复杂的实时调试。芯片中还集成了 IEEE 1149.1 (JTAG) 边界扫描功能。

## 15.2 Analog Devices 公司 ADSP-BF535 Blackfin 处理器

ADSP-BF535 处理器是 Blackfin 处理器系列产品的成员之一，实现了微信号体系结构（Micro Signal Architecture, MSA），由 Analog Devices 公司和 Intel 公司联合开发。处理器的基本功能框图如图 15-22 所示。

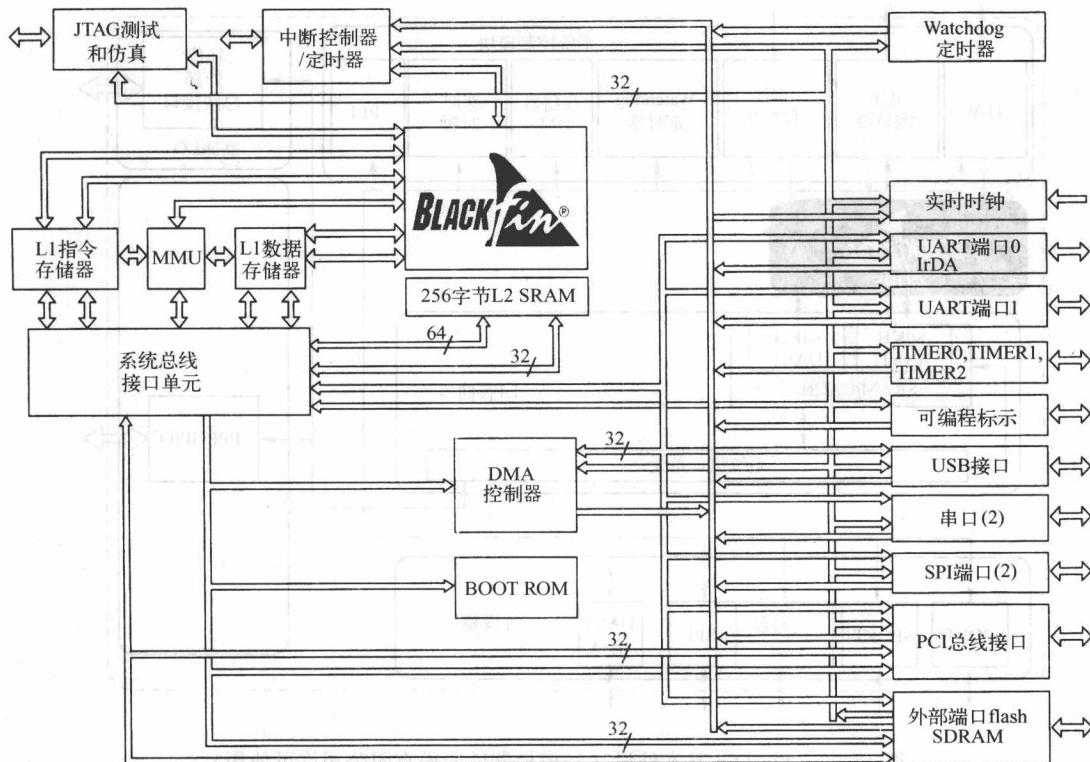


图 15-22 BF-535 功能框图 (经模拟器件股份有限公司许可使用)

体系结构在一个指令集体系结构中实现了众多的功能，包括一个双 MAC 先进信号处理引擎、类 RISC 微处理器体系结构、单指令多数据（SIMD）多媒体功能等。Blackfin 处理器中集成了丰富的工业系统外设，是下一代应用可以选择的平台。这些应用需要在一个集成块中包含类 RISC 的可编程性、多媒体支持、前沿信号处理。

### 15.2.1 便携低功耗体系结构

Blackfin 处理器具有先进的电源管理功能，采用低功耗、低电压设计技术设计，具有动态电源管理的特点，可以单独设置电源和工作频率，极大地降低了整个系统的功耗。电压和工作频率的改变可以显著降低功耗，进而延长便携式产品的电池寿命。

### 15.2.2 系统集成

ADSP-BF535 Blackfin 处理器是高度集成的片上系统，适用于下一代数字通信和便携式 Internet 设备。该处理器将工业标准接口与高性能信号处理内核相结合，可以加快用户的开发速度；由于不需要昂贵的外部设备，还可以降低成本。ADSP-BF535 Blackfin 处理器系统外设包括

UART、SPI、SPORT、通用定时器、实时时钟、可编程标示、Watchdog 定时器以及 USB 和 PCI 总线，用于外部设备的扩展。

ADSP-BF535 Blackfin 处理器的外设通过高带宽总线与内核相连（见图 15-23），使得系统配置更加灵活，提高了总体的系统性能。基本的外设包括通用功能，例如 UART、带有 PWM（脉宽调制）和脉冲计量功能的定时器、通用标示 I/O 引脚、一个实时时钟和一个 Watchdog 定时器。

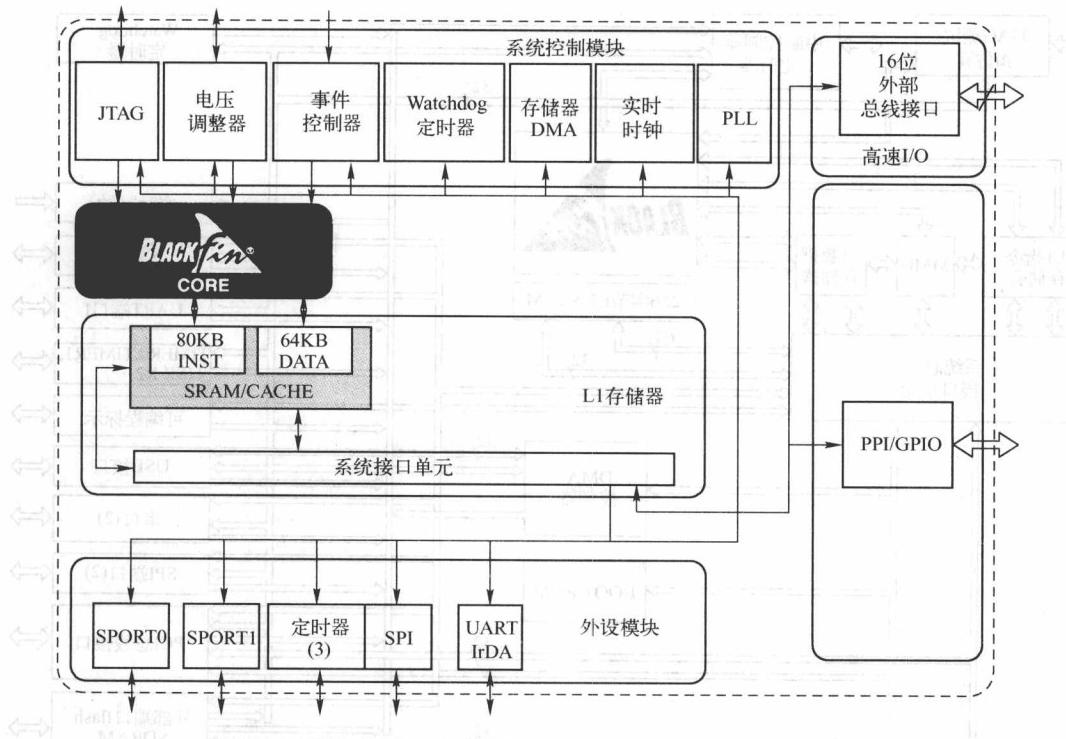


图 15-23 Blackfin 基本外设（经模拟器件股份有限公司许可使用）

这组功能能够满足各种典型系统的需求，并且还可以通过系统扩展功能进行补充。除了这些通用外设之外，ADSP-BF535 处理器还有一个高速串行端口，可以与各种音频和 modem CODEC 功能接口。还有一个事件处理器，可以对来自片上外设和外部中断进行灵活管理。芯片中还有一个电源管理控制功能，可以针对各种应用情形，对处理器和系统的性能、功耗进行不同的设置。

在许多系统设计中，仅用很少甚至不用粘胶逻辑，就可以很容易地添加片上外设，因为芯片中设置了各种接口，可以对工业标准总线进行扩展，这些接口包括一个 32 位、33 MHz、V2.2 兼容的 PCI 总线、SPI 串行扩展端口和一个设备类 USB 端口。这样可以根据不同的应用，将各种外设连接到系统中，而设计复杂性不会很大。

除可编程标示、实时时钟和定时器外，所有外设都由灵活的 DMA 结构提供支持，每个外设中都集成了单独的 DMA 通道。此外，还有一个独立的存储器 DMA 控制器，专门用于在各种存储器空间之间传输数据，包括外部 SDRAM 和异步存储器、内部 1 级和 2 级 SRAM 和 PCI 存储空间。

芯片中有多条 32 位的总线，运行速度最高可达 133MHz，为处理器核和所有的片上和外部设备同时运行提供了足够的带宽。

### 15.2.3 处理器核

如图 15-24 所示，Blackfin 处理器核包含两个乘法器/累加器（MAC）、两个 40 位的 ALU，4

个视频 ALU 和一个移位器。计算单元对寄存器文件中的 8 位、16 位或 32 位数据进行处理。每个 MAC 在一个周期中可以执行一个  $16 \text{ 位} \times 16 \text{ 位}$  的乘法运算，并且对 40 位的结果进行累加，扩充精度为 8 位。

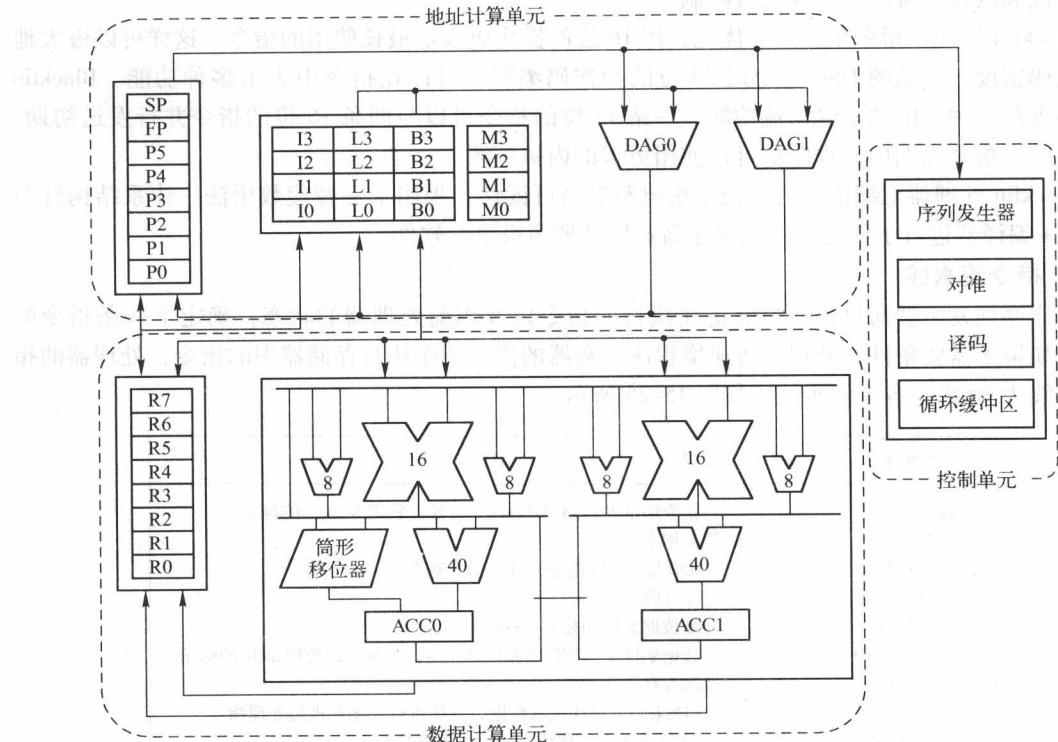


图 15-24 ADSP-BF533 处理器核体系结构（经模拟器件股份有限公司许可使用）

ALU 可以执行一组标准的算术和逻辑运算。芯片中有 2 个 ALU，可以处理 16 位或 32 位的数据，这种计算单元的灵活性满足了信号处理中各种应用的需求。2 个 32 位输入寄存器中的任何一个都可以分解为 2 个 16 位数据，因此任何一个 ALU 都可以完成非常灵活的 16 位算术运算。将寄存器看做 2 个 16 位操作数对，可以在一个周期中完成 2 个 16 位或 1 个 32 位的运算。利用第二个 ALU，可以很容易完成双 16 位操作。这样可以提高单周期的吞吐量。

40 位移位器的功能很强大，可以执行数据的移位、旋转、规格化、抽取和发送。计算单元的数据来自一个多端口的寄存器文件，具有 16 个 16 位的入口或 8 个 32 位的项。程序序列发生器控制着指令流的执行，还具有指令对准和译码功能。序列发生器支持条件跳转和子程序调用功能，并能实现零开销循环。循环缓冲区在本地存储指令，在循环的执行过程中不需要访存。

两个数据地址发生器（data address generator, DAG），可以为同时从存储器中读取的两个操作数提供地址。两个 DAG 共享一个寄存器文件，包含 4 组 32 位的 index、修改、长度和基址寄存器；还有 8 个 32 位的寄存器，为变量和堆栈地址的索引提供指针。

Blackfin 处理器支持 Harvard 体系结构，并结合了层次存储器结构。1 级（L1）存储器的操作速度与处理器相匹配，只有很少甚至没有延迟。2 级（L2）存储器可以在片上，也可以在片外，需要几个处理器周期对其进行访问。在 L1 中，指令存储器只保存指令。

两个数据存储器保存数据，还有一个专用的 scratch pad 数据存储器保存堆栈和局部变量信息。在 L2 中只有一个联合的存储空间，可以被配置为晶体 RAM（SRAM）或者 cache。

存储器管理单元 (MMU) 为每个运行在内核中的任务提供存储保护，并且保护系统寄存器不接受未经允许的读写。体系结构提供了 3 种操作模式，即用户模式、管理模式和仿真模式。

用户模式对于一些系统资源的访问受限，因此可以提供一个受保护的软件环境，而管理模式对系统和内核的资源访问都没有限制。

Blackfin 处理器指令集经过了优化，用 16 位的操作码表示最长使用的指令，这样可以极大地提高代码密度。复杂的 DSP 指令用 32 位的操作码编码，可以在指令中表示多种功能。Blackfin 处理器支持一种受限的多输出流功能，一条 32 位的指令可以与两条 16 位的指令并行发送初期，这样在一个指令周期中，程序员可以使用更多的内核资源。

Blackfin 处理器汇编语言为了便于编码和提高可读性，采用了一种代数语法。体系结构针对 C/C++ 编译器进行了优化，可以快速高效地处理高级语言软件。

### 1. 指令流水线

程序序列发生器通过检查当前正在执行的指令以及当前处理器的状态，确定下一条指令的地址。如果不需要条件，处理器将递增程序计数器的值，顺序执行存储器中的指令，处理器的指令流水线为 10 级，各个流水线级如图 15-25 所示。

流水线站	描述
取指 1 (IF1)	将指令地址发送到 IAB 总线，开始比较指令 cache 的标示
取指 2 (IF2)	等待指令
取指 3 (IF3)	从 IDB 总线读取指令，进行指令对准
指令译码 (DEC)	指令译码
地址计算 (AC)	计算数据地址和分支目标地址
取数据 1 (DF1)	将数据地址发送到 DA0 和 DA1 总线，开始比较数据 cache 的标示
取数据 2 (DF2)	读取寄存器文件
执行 1 (EX1)	从 LD0 和 LD1 总线读取数据，开始执行乘法和视频处理指令
执行 2 (EX2)	执行/结束指令（移位、加法、逻辑等）
写回 (WB)	将结果写回寄存器文件、SD 总线，并进行指针更新（也称为“结束”站）

图 15-25 ADSP-BF533 指令流水线站（经模拟器件股份有限公司许可使用）

### 2. 指令流水线指令流

图 15-26 给出了处理器流水线的示意图。取指和分支逻辑产生 32 位的取指地址，用于访问指令存储器单元。指令对准单元在 IF3 站的最后，返回指令以及指令宽度信息。对于每一种指令类型（16 位、32 位或 64 位），指令对准单元都要确保对准缓冲区中有足够的有效指令，确保在每一个周期中都向流水线提供一条指令。因为指令可以是 16、32 或 64 位的，指令对准单元可以不必每个周期都从 cache 中读取指令。

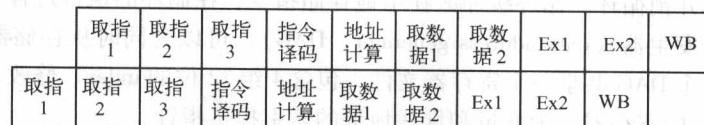


图 15-26 ADSP-BF533 处理器流水线（经模拟器件股份有限公司许可使用）

对于一个 16 位的指令序列，指令对准单元每 4 个周期从指令存储单元中读取一条指令。对准逻辑根据对准缓冲区的状态请求下一条指令的地址，如果程序流不发生变化，序列发生器做出响应，在下一个周期产生下一个取指地址；序列发生器将一直保存取指地址，直到它收到来自对准逻

辑的请求，或者发生了程序流的变化为止。序列发生器总是将前一个取指地址加8（下一条8字节指令），如果发生了指令流的变化，例如分支或中断，指令对准单元中的数据将视为无效；序列发生器将对指令字进行译码，并将指令数据发送到适当的地方，如寄存器文件和数据存储器。

### 15.2.4 存储器体系结构

ADSP-BF535 Blackfin处理器将存储器（见图15-27）看做一个联合的4G字节寻址空间，地址为32位。所有的资源，包括内部存储器、外部存储器、PCI地址空间和I/O控制寄存器，都是这个寻址空间的一部分。该地址空间的存储空间按照层次结构进行组织，可以实现很好的成本/性能平衡，有与处理器离得最近的快速、低延迟存储器，如cache或SRAM，还有距离处理器较远的面积较大、成本较低、性能较差的存储器系统。

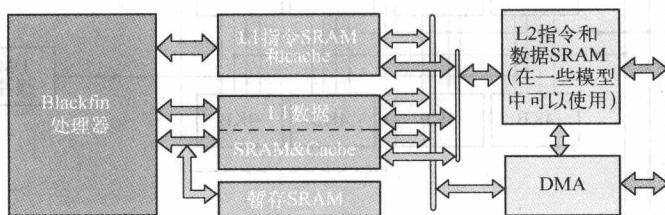


图15-27 Blackfin存储器体系结构（经模拟器件股份有限公司许可使用）

L1存储系统是Blackfin处理器核性能最高的存储器。该存储系统在片外，通过外部总线接口单元（external bus interface unit, EBIU）对它进行访问，扩展了SDRAM、flash存储器和SRAM，可以有选择地访问至少768M字节的外部物理存储器。

存储器DMA控制器具有宽带数据移动功能。它可以在内部L1/L2存储器和外部存储空间（包括PCI存储空间）之间进行代码块或数据块的传输。

#### 1. 内部（片上）存储器

ADSP-BF535 Blackfin处理器有4个片上存储器块，可以提供高带宽的内核访问。首先是L1指令存储器，由一个16字节的4路组相联cache存储器构成，如图15-28所示。此外，存储器还可以被配置为SRAM，这个存储器可以按照处理器的速度访问。

第二个片上存储器块是L1数据存储器（见图15-27），由两个16K字节的块构成。每一个L1数据存储器块都可以配置为两路组相联cache中的一路，也可以配置为一个SRAM，可以按照处理器核的速度访问。

第三个存储块是一个4K字节的暂存内存，与L1存储器的运行速度相同；但是只能作为数据SRAM使用，不能被配置为cache存储器，不能通过DAM访问。

第四个片上存储系统是L2 SRAM存储阵列，有256K字节的高速SRAM，带宽与内核相同，比L1存储块的延迟长。L2存储器既可以存放数据，也可以存放指令；还可以根据系统设计的需要，混合存储指令和数据。Blackfin处理器核有一个专门的低延迟64位宽的数据通路，用于访问L2 SRAM存储器。

#### 2. PCI

PCI总线定义了3个独立的地址空间，通过ADSP-BF535 Blackfin处理器存储器空间进行访问。这些存储器空间是PCI存储器、PCI I/O和PCI配置空间。此外PCI接口可以用作处理器核的桥，在系统中作为控制CPU，或者用作主端口；由系统中的另外一个CPU作为主设备，ADSP-BF535作为PCI总线上的一个智能I/O设备。

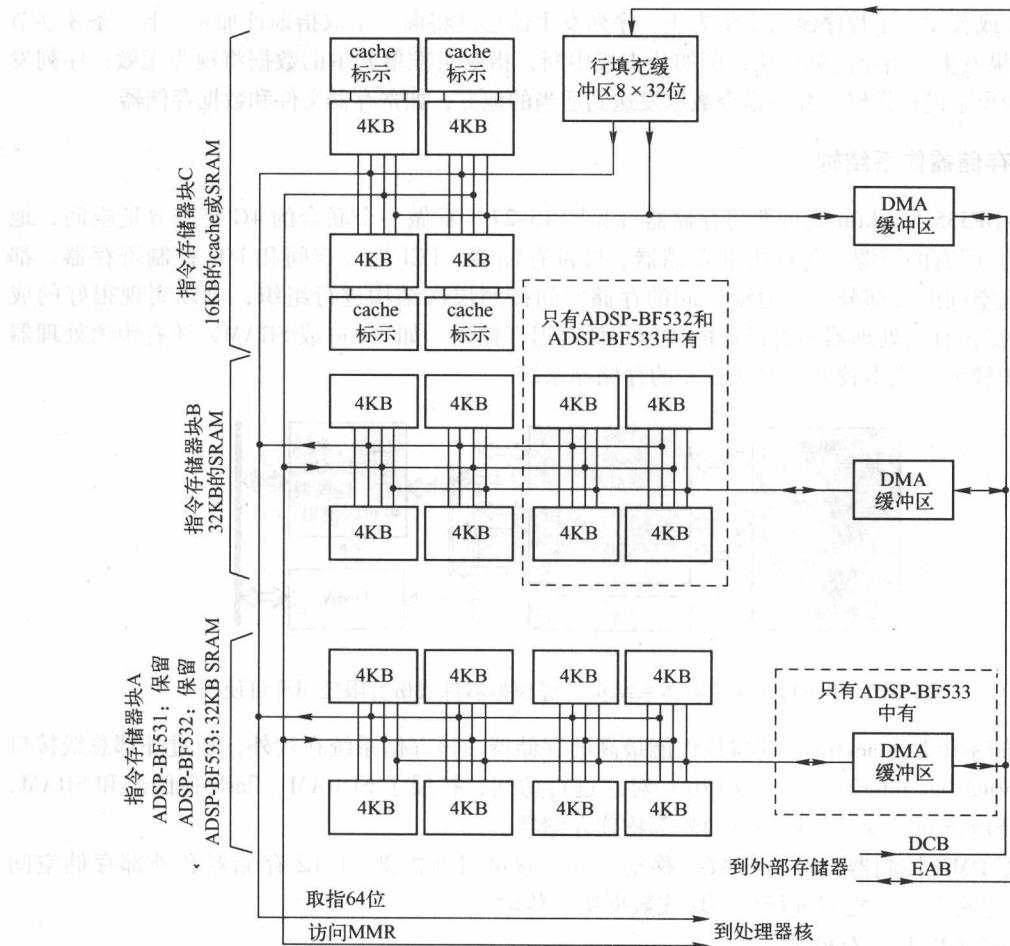


图 15-28 ADSP-BF533 L1 指令存储器体系结构（经模拟器件股份有限公司许可使用）

当 ADSP-BF535 Blackfin 处理器用作系统控制器时，它可以通过映像窗口对 PCI 地址空间进行访问，可以使用系统中的所有设备，并对系统环境进行总体控制。PCI 存储空间是 4K 字节，位于 PCI 总线上，可以用于映像总线上的存储器 I/O 设备。

ADSP-BF535 Blackfin 处理器在存储空间中设置了一个 128 M 字节的窗口，可以访问 PCI 存储空间的一部分。基地址寄存器对该窗口进行定位，窗口可以位于 4G 字节 PCI 存储空间的任何位置，而它相对于处理器地址的位置却是固定的。

PCI I/O 空间也是一个 4G 字节的空间。然而，大多数的系统和 I/O 设备只使用这个空间中 64 K 字节的子集，用作 I/O 映射的地址。ADSP-BF535 Blackfin 处理器中设计了一个 64K 字节的窗口，还有一个与其相关的基地址寄存器，基地址寄存器可以将该窗口定位在 PCI I/O 寻址空间的任何位置，而窗口本身的地址在处理器的地址空间中保持不变。

PCI 配置空间是一个有限的地址空间，用于系统枚举和初始化，该地址空间是处理器和 PCI 设备之间的低速通信通道。ADSP-BF535 Blackfin 处理器中有一个单值窗口，用于访问 PCI 配置空间中任何地址上的数据值。这个窗口是固定的，接收数值的地址；在写操作情况下，接收数据值本身；而在读操作时，该设备将数值返回到相同的地址中。

### 3. I/O 存储空间

Blackfin 处理器没有定义单独的 I/O 空间。所有资源都通过 32 位的地址进行映射。片上 I/O 设备在存储器映射寄存器 (memory-mapped register, MMR) 中，都有自己的控制寄存器，MMR 的地址位于 4G 字节寻址空间的顶部。这些寄存器分为两个小块，一个块中是所有内核功能的控制 MMR，另一块中的寄存器用于设置和控制内核以外的片上外设。

内核 MMR 只能由内核在管理模式下访问，是片上外设的保留空间，也是可以通过 PCI 总线访问的外部设备资源。系统 MMR 可以由内核在管理模式下访问，根据系统的保护模式，对其他设备可以是可见的，也可以是保留的。

#### 15.2.5 事件处理

ADSP-BF535 Blackfin 处理器中的事件控制器处理所有的异步和同步事件。ADSP-BF535 Blackfin 处理器既支持嵌套的事件处理，也支持优先级的事件处理。嵌套模式允许多个事件服务子程序同时运行。

优先级方式确保具有高优先级的事件可以在低优先级事件之前进行处理。控制器提供了如下 5 种不同事件类型的处理。

- 仿真 仿真事件将使处理器进入仿真模式，可以通过 JTAG 接口对处理器进行控制。
- 复位 该事件将使处理器复位。
- 非屏蔽中断 (nonmaskable interrupt, NMI) NMI 事件可以由软件 Watchdog 定时器触发，或者由处理器的 NMI 输入信号触发。NMI 事件通常用作节电指示器，使系统有序关机。
- 异常 与程序流的执行同步发生的事件，例如，异常在指令执行完之前产生。会产生异常的情况包括数据对准违反、未定义的指令等。
- 中断 与程序流的执行异步发生的事件，该事件可以由定时器、外设、输入引脚、特定的软件指令等触发。

每一个事件都有一个相关的寄存器，用于保存返回地址，还有一个相关的事件返回指令。当一个事件被触发之后，处理器的状态保存在管理堆栈中。

ADSP-BF535 Blackfin 处理器的事件控制器由两部分组成，即内核事件控制器 (core event controller, CEC) 和系统中断控制器 (system interrupt controller, SIC)。内核事件控制器与系统中断控制器共同对所有的系统事件进行优先化处理和控制。从概念上讲，来自外设的中断将进入 SCI，然后直接发送到 CEC 的通用中断。

#### 1. 内核事件控制器 (CEC)

除了专用的中断和异常事件之外，CEC 支持 9 个通用中断 (IVG15-7)。在这些通用中断中，两个优先级最低的中断 (IVG15-14) 保留给软件中断控制器使用，另外 7 个优先级较高的中断输入支持 ADSP-BF535 Blackfin 处理器的外设。

图 15-29 描述了 CEC 的输入，给出了它们在事件向量表 (Event Vector Table, EVT) 中的名称，列出了它们的优先级。

#### 2. 系统中断控制器 (SIC)

SIC 将来自外设中断源的事件，映射和发送到优先级控制的通用中断 CEC 的输入。尽管 ADSP-BF535 Blackfin 处理器中有默认的映射，用户也可以通过改写中断分配寄存器 (interrupt assignment register, IAR) 中的值，对中断事件的映射和优先级进行修改。图 15-30 描述了 SIC 的输入以及向 CEC 的默认映射。

优先级 (0 为最高)	事件类别	时间入口
0	仿真测试	EMU
1	复位	RST
2	不开屏蔽	NMI
3	异常	EVX
4	全局使能	
5	硬件故障	IVHWF
6	内核定时器	IVTMR
7	通用中断 7	IVG7
8	通用中断 8	IVG8
9	通用中断 9	IVG9
10	通用中断 10	IVG10
11	通用中断 11	IVG11
12	通用中断 12	IVG12
13	通用中断 13	IVG13
14	通用中断 14	IVG14
15	通用中断 15	IVG15

图 15-29 ADSP-BF533 内核事件控制器 (经模拟器件股份有限公司许可使用)

外设中断事件	外设中断ID	缺省映射
实时时钟	0	IVG7
保留	1	
USB	2	IVG7
PCI 中断	3	IVG7
SPORT 0 Rx DMA	4	IVG8
SPORT 0 Tx DMA	5	IVG8
SPORT 1 Rx DMA	6	IVG8
SPORT 1 Tx DMA	7	IVG8
SPI 0 DMA	8	IVG9
SPI 1 DMA	9	IVG9
UART 0 Rx	10	IVG10
UART 0 Tx	11	IVG10
UART 1 Rx	12	IVG10
UART 1 Tx	13	IVG10
定时器0	14	IVG11
定时器1	15	IVG11
定时器2	16	IVG11
GPIO 中断A	17	IVG12
GPIO 中断A	18	IVG12
存储器 DMA	19	IVG13
软件 watchdog 定时器	20	IVG13
保留	26-21	
软件中断1	27	IVG14
软件中断2	28	IVG15

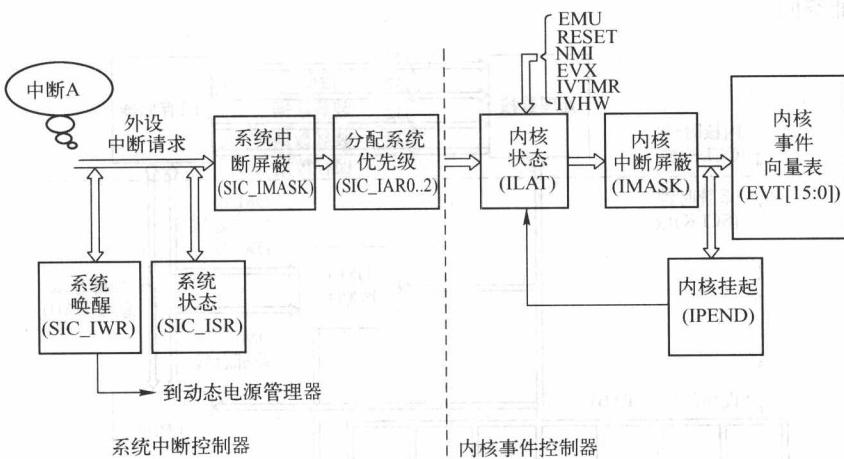
图 15-30 ADSP-BF533 系统中断控制器 (经模拟器件股份有限公司许可使用)

### 3. 中断事件控制

中断是一个事件，它改变了处理器的正常指令流，与程序流异步发生；与之相反，异常是由软件触发的，是与程序流同步发生的。事件系统可以嵌套，并设置优先级。因此，在任何时间都可以发生多个服务子程序，低优先级的事件将被高优先级的事件抢先执行。

处理器采用了一种两级事件控制机制，处理器 SIC 与 CEC 一起对所有的系统中断进行优先化和控制。SIC 将多个外设中断源映射到内核优先级控制的通用中断输入，这种映射是可编程的，在 SIC 中可以对中断源进行单独屏蔽。

ADSP-BF535 Blackfin 处理器为用户提供了一种非常灵活的机制，来控制事件的处理。基本的中断处理流程框图如图 15-31 所示。



注：圆括号中的名称是存储器映射的寄存器。

图 15-31 ADSP-BF533 中断处理框图（经模拟器件股份有限公司许可使用）

在 CEC 中有 3 个寄存器用于事件的协调和控制，每一个寄存器都是 16 位的，每一个位都代表了一个特定的事件类别。SIC 提供了 3 个 32 位的控制和状态寄存器，进一步控制事件的处理。在寄存器中，每一个外设中断事件都有一个位与之对应。

因为多个中断源可以被映射到一个通用中断上，因此，在对已经在该中断输入上检测到的中断事件进行处理之前或处理过程中，可能会同时发送出多个中断请求。SIC 将监视 IPEND 寄存器中的内容，进行中断响应。

当检测到中断上升沿后（检测需要两个内核时钟周期），ILAT 寄存器中的相应位被设置。当对应的 IPEND 寄存器位被设置后，ILAT 寄存器中的位将被清除。IPEND 位表示事件已经进入了处理器流水线。这时，CEC 识别相应事件输入上的下一个上升沿事件，并对这个事件进行排队。从通用中断的上升沿发生，到 IPEND 输出，最小延迟是 3 个内核时钟周期；然而，根据处理器正在执行的操作和状态，这个延迟可能会更长。

### 15.2.6 DMA 控制器

在 ADSP-BF535 Blackfin 处理器的内部存储器和任何一个有 DMA 功能的外设之间，都可以进行 DMA 传输。此外，DMA 传输还可能发生在任何一个有 DMA 功能的外设和连接到外部存储器接口上的外部设备之间，包括 SDRAM 控制器、异步存储控制器和 PCI 总线接口，如图 15-32 所示。

有 DMA 功能的外设包括 SPORT、SPI 端口、UART 和 USB 端口。每一个有 DMA 功能的外设都至少有一个专用的 DMA 通道，进出 PCI 的 DMA 由存储器 DMA 通道实现。

DMA 控制器采用一组称为描述符块的参数，描述每一个 DAM 的序列。当需要连续 DMA 序列时，这些描述符块可以连接或链接到一起，这样一个 DMA 序列的结束将初始化和启动下一个序列。描述符块 32 位的地址，用作源和目的操作数的基址指针，可以对 ADSP-BF535 Blackfin 处理器的整个寻址空间进行访问。

除了专门的外设 DMA 通道外，还有一个单独的存储器 DMA 通道，用于在 ADSP-BF535

Blackfin 处理器系统的各个存储器之间进行数据传输。这样可以在处理器干预很少的情况下，在任何存储器之间传输数据块，包括片上 2 级存储，外部 SDRAM、ROM、SRAM 和 flash 存储器，以及 PCI 地址空间。

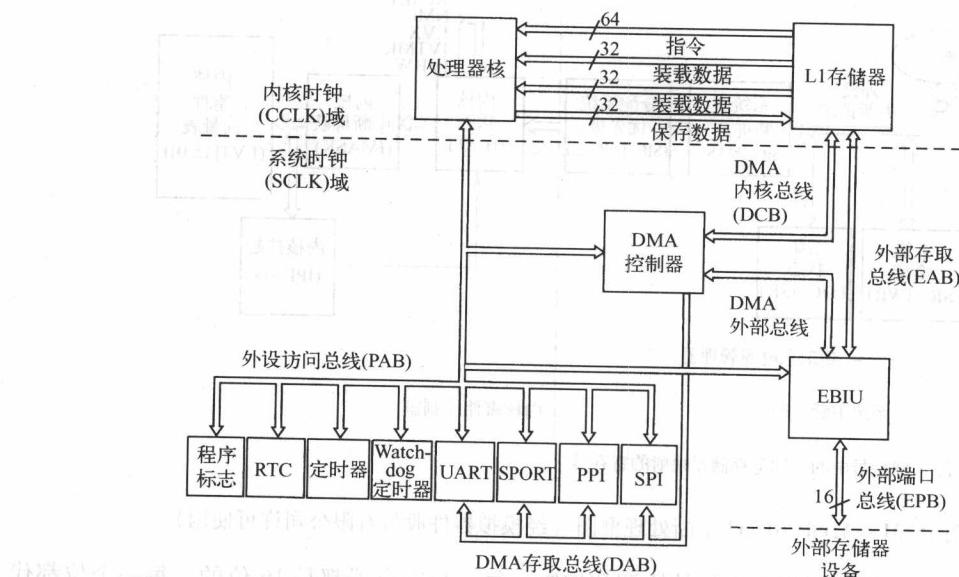


图 15-32 ADSP-BF533 DMA 控制器接口（经模拟器件股份有限公司许可使用）

### 15.2.7 外部存储控制

图 15-33 给出了 ADSP-BF535 Blackfin 处理器中的外部总线接口单元（EBIU）结构示意。它提供了一种与各种工业标准的存储器设备进行接口的高性能、无粘接接口。控制器由两部分组成：第一部分是 SDRAM 控制器，与工业标准同步 DRAM 设备和 DIMM（双列直插式内存模块）进行连接；第二部分是异步存储控制器，可以与各种存储器设备进行接口。

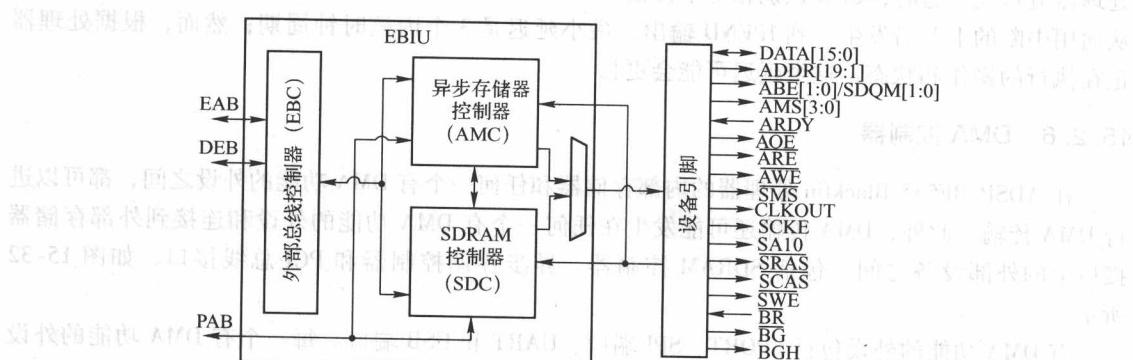


图 15-33 ADSP-BF533 外部总线接口单元（经模拟器件股份有限公司许可使用）

#### SDRAM 控制器

SDRAM 控制器最多可以与 4 块工业标准 SDRAM 器件或 DIMM 进行接口，最高速度可达  $f_{S-CLK}$ 。存储器块与 PCI133 SDRAM 标准完全兼容，每个存储器块都可以配置为空间大小为 16 ~

128 M 字节的存储器。控制器将所有块看做连续的地址空间，这样即使在不同的块中使用的存储器大小不同，处理器也可将整个空间看做一个地址空间。这样就允许系统采用相同或者不同类型的存储器进行配置升级。

通过一组可编程的时序参数，可以对 SDRAM 块进行配置，以支持慢速存储器设备。存储器块可以被配置为 32 位宽，获得最高的性能和带宽，或者配置为 16 位宽，以减小设备数量和降低系统成本。

4 个存储器块共享 SDRAM 控制信号，并有自己的块选择线，这样就为大多数系统配置提供了一个完全无粘胶接口。SDRAM 控制器的地址、数据、时钟和命令引脚可以驱动最大 50 pF 的负载。对于较大的存储器系统，可以选择 SDRAM 控制器外部缓冲区时序，提供外部缓冲区，使 SDRAM 控制器引脚上的负载不超过 50 pF。

### 15.2.8 异步控制器

异步存储控制器为多达 4 个分离的内存条或 I/O 设备提供了一个可配置的接口。每一个内存条都可以用不同的时序参数进行独立编程，使得控制可以与各种类型的存储设备进行连接，包括 SRAM、ROM 和 Flash EEPROM，以及那些与标准存储控制器接口的 I/O 设备。

每一个内存条在处理器的地址空间中占用一个 64M 字节的窗口，但是如果不是完全板上组装的，存储控制器逻辑并不会使这些窗口相邻。内存条也可以被配置为 16 位宽或 32 位宽的总线，易于与一定范围的存储器和 I/O 设备进行接口，这些存储器和 I/O 设备的设置或者是为了高性能，或者是为了低成本和低功耗。

### 15.2.9 PCI 接口

ADSP-BF535 Blackfin 处理器提供了一个 33 MHz、3.3 V、32 位、与 2.2 版本兼容的无粘胶逻辑和 PCI（外设组件互联）接口。PCI 接口可以用于 3V 的电压环境中，提供了处理器核和片上外设以及外部 PCI 总线之间的总线桥功能。ADSP-BF535 Blackfin 处理器的 PCI 接口支持如下两种 PCI 功能：

- PCI 桥主设备功能。从 PCI 目标设备来看，ADSP-BF535 Blackfin 处理器的资源（处理器核、内部和外部存储器核、存储器 DMA 控制器）是必要的硬件组成部分，用以模拟主计算机 PCI 接口。
- PCI 目标设备功能。ADSP-BF535 Blackfin 基于处理器的智能外设可以与 2.2 版本的 PCI 总线连接。

#### 1. PCI 主设备功能

作为 PCI 主设备时，ADSP-BF535 Blackfin 处理器提供必要的 PCI 主设备（平台）功能，支持和控制各种市场销售的 PCI I/O 设备。这些设备可以是在 ADSP-BF535 Blackfin 处理器作为主设备的系统中的 Ethernet 控制器、总线桥等。

注意，Blackfin 处理器体系结构只定义了内存空间（没有 I/O 或者配置地址空间）。PCI 空间的 3 个地址空间（内存、I/O 和配置空间）都被映射到了 ADSP-BF535 Blackfin 处理器的平面 32 位内存空间。

由于 PCI 内存空间与 ADSP-BF535 Blackfin 处理器的内存地址空间一样大，因此采用了分窗口的方法，ADSP-BF535 Blackfin 处理器地址空间中有分离的窗口，用于访问 3 个 PCI 地址空间。还提供了基址寄存器，使得这些窗口可以被定位到查看 PCI 地址空间中的任何范围，而同时窗口在 ADSP-BF535 Blackfin 处理器的地址范围内保持固定位置。

为了使 PCI 总线上的设备能够查看 ADSP-BF535 Blackfin 处理器的资源，还提供了许多映射寄存器，使得可以在 PCI 地址空间中查看这些资源。ADSP-BF535 Blackfin 处理器的外部内存资源、内部 L2 和一些 I/O MMR 可以被有选择地使能为内存空间，PCI 总线上的设备可以用作 PCI 内存事务的目标。

### 15.2.10 USB 设备

ADSP-BF535 Blackfin 处理器中有一个 USB 兼容的设备接口，支持与主系统的直接连接。USB 内核接口提供了一个灵活的可编程环境，最多可以支持 8 个端点。每一个端点都可以支持所有的 USB 数据类型，包括控制、块、中断同步。每一个端点都有一个存储器映射缓冲区，用于向应用传输数据。

ADSP-BF535 Blackfin 处理器 USB 接口有一个专用的 DMA 控制器和一个中断输入，减小处理器的查询消耗；当需要传输管理时，可以异步请求 CPU 的处理。USB 设备需要一个外部的 48 MHz 的振荡器。为了保证 USB 的正确操作，SCLK 的频率必须超过 48 MHz。

### 15.2.11 实时时钟

ADSP-BF535 Blackfin 处理器实时时钟（real-time clock，RTC）提供了一组强大的数字计时功能，包括当前时间、秒表和闹钟，如图 15-34 所示。RTC 由 ADSP-BF535 Blackfin 处理器外部的 32.768 kHz 的晶振提供振源。RTC 外设有专用的电源引脚，这样即使处理器的其他部分处于低功耗状态，也可以保证电源和时钟的提供。RTC 有许多可编程的中断选项，包括每秒、每分或计时时钟中断，根据可编程的读秒定时中断，或者根据可编程的闹钟中断。

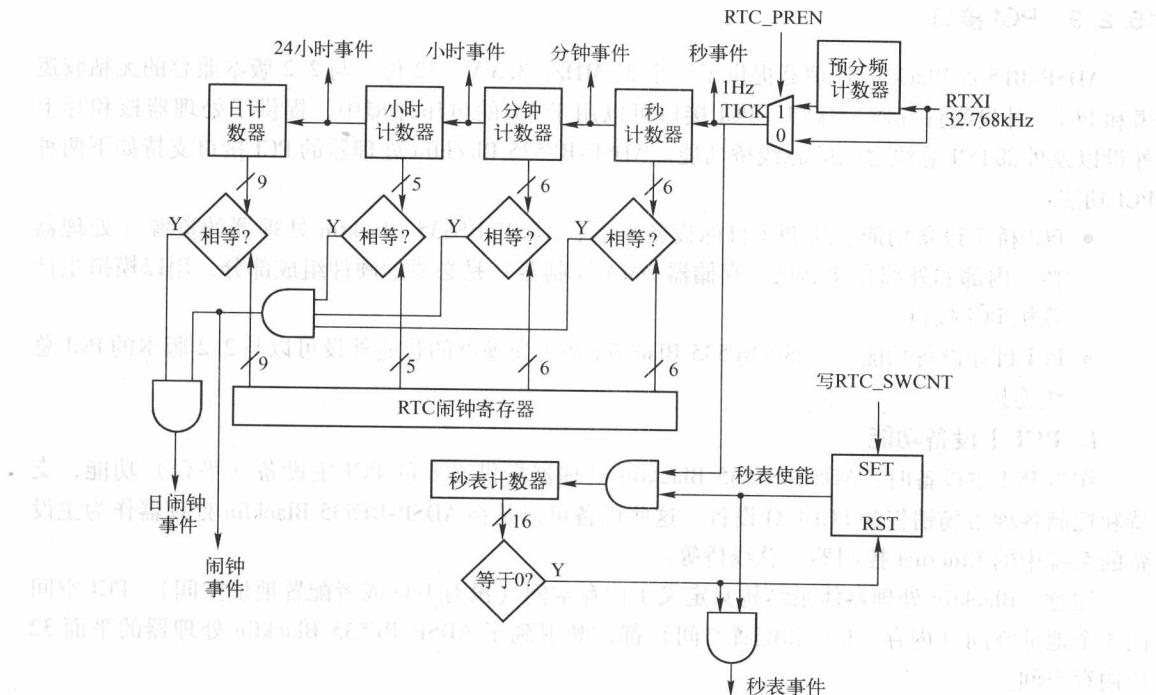


图 15-34 ADSP-BF533 实时时钟（经模拟器件股份有限公司许可使用）

32.768 kHz 的输入时钟频率可以由预分频器分频为 1 Hz。定时器的计数器功能有 4 种计数器——一个 6 位的秒计数器、一个 6 位的分计数器、一个 5 位的小时计数器和一个 8 位的日计数器。

如果实时时钟被启用，当定时器的输出与闹钟控制寄存器中预先编程的值相匹配时，闹钟功能将产生一个中断；共有两个闹钟：一个用于时刻，一个用于天以及当天的时间。

秒表功能从一个设定的值开始倒计时，精度为1分钟。当秒表功能被启用，并且计数器下溢时，就会产生中断。与其他外设类似，发生任何中断，RTC都可以将ADSP-BF535 Blackfin处理器从低功耗状态唤醒。

### 15.2.12 Watchdog定时器

ADSP-BF535 Blackfin处理器中有一个32位的定时器，可以用于实现软件Watchdog功能。如果在被软件复位之前发生了定时器终止，软件Watchdog可以将处理器设置为某个状态，提高系统的可用性，这种状态设置可以通过产生硬件复位、不可屏蔽中断（NMI）或者通用中断来实现。程序员对定时器的计数值进行初始化，将中断设置为使能，就可以使用定时器了。此后，在定时器从设定的值开始计数，到达零值之前，软件必须对计数器进行重新装载，避免系统保持在一个未知的状态。在该状态下，能够对定时器进行正常复位的软件已经由于外部干扰条件或软件故障而被停止。

被复位之后，软件将通过查询定时器控制寄存器中的状态位确定Watchdog是否硬件复位的原因，该状态位只有在Watchdog发生复位时才会被设置。定时器由系统时钟（SCLK）提供时钟源，最高频率为fsCLK。

### 15.2.13 定时器

在ADSP-BF535 Blackfin处理器中共有4个可编程定时器单元，图15-35给出了定时器功能的基本框图。3个通用定时器有一个外部引脚，可以被配置为脉宽调制器（PWM）或者定时器输出，作为定时器的输入时钟源，或者检测外部时间的脉冲宽度。

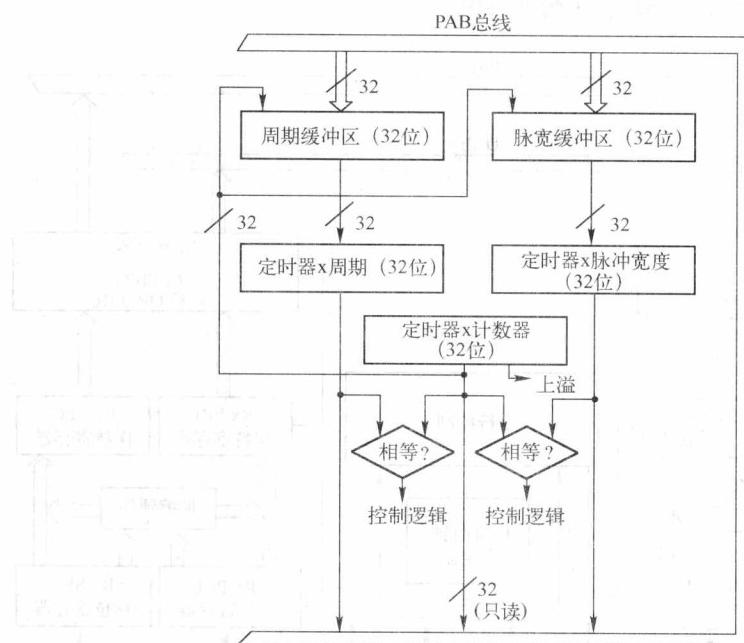


图15-35 ADSP-BF533通用定时器功能框图（经模拟器件股份有限公司许可使用）

3 个通用定时器中的每一个都可以单独编程为 PWM、内部或外部提供时钟的定时器或者脉宽计数器。通用定时器可以与 UART 相结合，来检测数据流中的脉冲宽度，为串行通道提供自动波特率检测功能。

通用定时器可以向处理器内核产生中断，提供与处理器时钟或者外部信号的计数同步的周期事件。除 3 个通用可编程定时器之外，处理器还提供了第 4 个定时器。这个定时器由内部处理器时钟（CCLK）作为时钟源，一般用作系统时钟，用于产生操作系统的周期性中断。

### 15.2.14 串口

ADSP-BF535 Blackfin 处理器中有 2 个完全同步的串口（SPORT0 和 SPORT1），用于串行和多处理器通信，是基本外设功能的一部分。

SPORT 框图如图 15-36 所示，有下列特性：

- 双向操作 每一个 SPORT 都有单独的发送和接收引脚。
- 具有缓冲功能的（深度为 8）发送和接收端口 每一个端口都有一个数据寄存器，在其他处理器组成部件之间传输数据；还包括可以对数据寄存器的值进行移位的移位寄存器。
- 时钟 每一个发送和接收端口都可以使用外部时钟，也可以产生自己的时钟，频率范围为  $(f_{SCLK}/131070)$  Hz ~  $(f_{SCLK}/2)$  Hz 之间。
- 字长 每一个 SPORT 支持 3 ~ 16 位的串行数据字宽度，传输格式为最高位优先或最低位优先。
- 分帧 每一个发送和接收端口在运行时，可以针对每个字设置帧同步信号，也可以不设置帧同步信号。帧同步信号可以由内部产生，也可以由外部产生；可以是低有效，也可以是高有效；脉冲宽度可以二选一，采用帧起点或帧终点同步。
- 硬件伸缩 根据 ITU 建议 G.711。每一个 SPORT 可以执行 A 律或 μ 律伸缩，伸缩可以在发送和/或接收通道进行选择，不需要延迟。

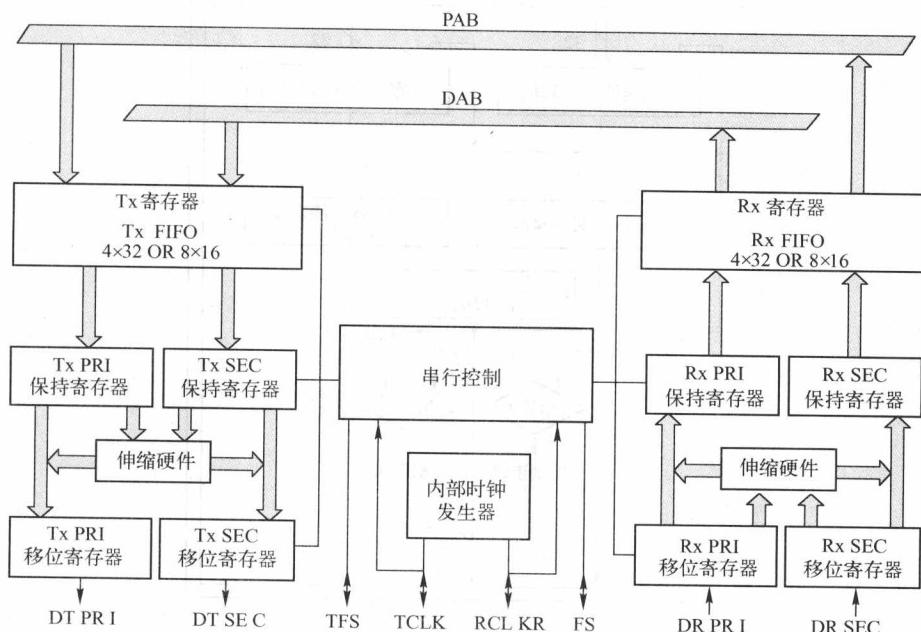


图 15-36 ADSP-BF533 SPORT 框图（经模拟器件股份有限公司许可使用）

- 单周期开销的 DMA 操作 每一个 SPORT 都可以自动接收和发送多个存储器缓冲区的数据。Blackfin 处理器将 SPORT 和存储器之间的 DMA 传输序列进行连接或链接，通过建立链接的描述符块，可以自动分配和更新链接的 DMA 序列。
- 中断 当通过 DMA 结束一个数据字的传输，或者完成一个或几个数据缓冲区的传输时，接收和发送端口将产生一个中断。
- 多通道功能 每一个 SPORT 支持 128 个通道，可以与 H.100、H.110、MVIP-90 和 HMVIP 标准相兼容。

### 15.2.15 串行外设接口 (SPI) 端口

ADSP-BF535 Blackfin 处理器有两个 SPI 兼容端口（见图 15-37），可以使处理器与多个 SPI 兼容的设备进行通信。SPI 接口在传输数据时使用 3 个引脚——两个数据引脚（主输出从输入、MOSIx，和主输入从输出、MISOx）和一个时钟引脚（串行时钟 SCKx）。两个 SPI 芯片选择输入引脚（SPISSx）供其他 SPI 设备选择处理器，14 个 SPI 芯片选择输出信号（SPIxSEL7-1）供处理器选择其他 SPI 设备。

SPI 选择引脚是可重配置可编程的标志引脚。使用这些引脚，SPI 端口提供了一种全双工、同步串行接口，该接口同时支持主模式和从模式以及多主设备环境。每个 SPI 端口的波特率和时钟相位/极性都是可编程的，并且每一个端口都有一个集成的 DMA 控制器，可以进行配置，支持数据流的发送和接收。SPI 的 DMA 控制器在任何时间，都只能支持单向访问。

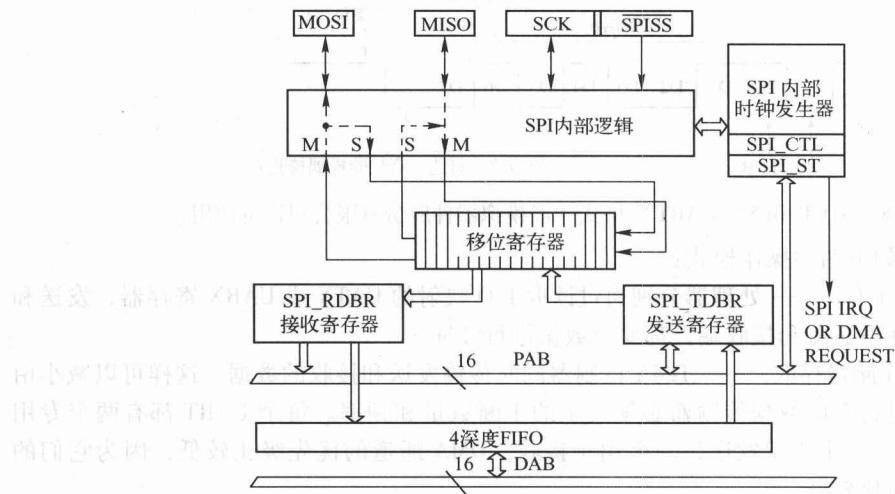


图 15-37 ADSP-BF533 SPI 框图（经模拟器件股份有限公司许可使用）

在传输期间，SPI 端口通过串行移位数据，同时对两条串行数据线上的数据进行发送和接收，串行时钟线对两条串行数据线上数据的移位和采样进行同步。在主模式中，处理器设置和初始化 SPI 传输的步骤如下。

- 1) 启用和配置 SPI 端口的操作（数据大小和传输模式）。
- 2) 选择目标 SPI 从设备，以及相关的 SPIxSELy 输出引脚（重配置可编程标志引脚）。
- 3) 在处理器的内存空间中定义一个或多个 TCB（只有在 DMA 模式下可选）。
- 4) 使能 SPI DMA 引擎，并描述传输的方向（只有在 DMA 模式下可选）。
- 5) 读或写 SPI 端口接收或发送数据缓冲区（只有在非 DMA 模式中可选）。



SCK<sub>x</sub> 线产生经过编程的时钟脉冲，可以同时提供给在 MOSI<sub>x</sub> 上的数据移出，和在 MISO<sub>x</sub> 上的数据移入。只有在 DMA 模式中，传输过程会一直进行，直到 SPI DMA 字数从 1 转换为 0 为止。

在从模式中，处理器设置 SPI 端口，并从一个主发送设备接收数据的步骤如下：

- 使能和配置 SPI 从端口，满足主 SPI 发送设备中所设置的操作参数（数据大小和传输模式）。
- 在处理器的内存空间中定义一个或多个 TCB，在数据传输结束时产生中断（只有在 DMA 模式下可选）。
- 使能 SPI DMA 引擎，准备接收数据（只有在 DMA 模式下可选）。
- 在从主设备接收 SPISS<sub>x</sub> 输入引脚上（重配置可编程标志引脚）的 SPI 芯片选择信号后，相应的 SPI SCK<sub>x</sub> 有效沿将开始接收数据。

通过将下一个命令 TCB 进行排队，处理器可以继续执行。从模式的发送过程类似，但是此时处理器将指定内存中发送数据的数据缓冲区，产生并放弃发送 TCB 的控制，并开始填充 SPI 端口的数据缓冲区。如果 SPI 控制器还没有准备好发送，它可以发送一个“零”字。

### 15.2.16 UART 端口

ADSP-BF535 Blackfin 处理器提供了两个全双工异步接收器传输总线 (UART) 端口 (UART0 和 UART2)，完全与 PC 标准的 UART 相兼容。UART 端口给其他外设或主设备提供了一个简化的 UART 接口，支持全双工、DMA 支持的异步串行数据传输。每一个 UART 端口可以支持 5~8 个数据位及 1 或 2 个停止位，支持偶校验或奇校验，或者没有校验 (见图 15-38)。

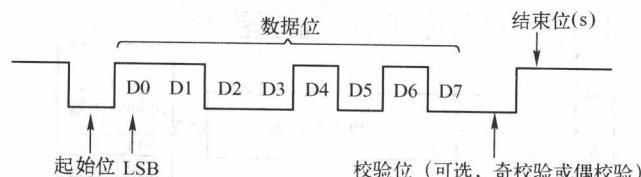


图 15-38 ADSP-BF533 UART 字格式 (经模拟器件股份有限公司许可使用)

UART 端口支持如下两种操作模式：

- PIO (编程的 I/O) —— 处理器分别通过读取 I/O 映射的 UATX 或 UARX 寄存器，发送和接收数据。在发送端和接收端，都要对数据进行缓冲。
- DMA (直接存储器存取) —— DMA 控制器同时传输发送和接收的数据。这样可以减小由于对存储器进行双向数据传输而必须发生的中断数量和频率。每个 UART 都有两个专用的 DMA 通道，一个用于发送，一个用于接收。DMA 通道的优先级比较低，因为它们的服务频率相对比较低。

每一个 UART 端口的波特率、串行数据格式、故障码产生和状态及中断，都是可编程的，指标如下：

- 位率范围为每秒 (FSCLK/1048576) ~ (FSCLK/16) 位。
- 数据格式 7~12 位每帧。
- 发送和接收操作可以进行配置，产生处理器的可屏蔽中断。

还支持自动波特率检测，与通用定时器功能一起使用；并对 UART0 的功能进行了进一步扩展，支持红外线数据协会 (IrDA) 串行红外线物理层规范 (SIR) 协议。

### 15.2.17 动态电源管理

ADSP-BF535 Blackfin 处理器提供了 4 种电源管理模式，每一种的性能/功耗都不同。此

外，动态电源管理还提供了控制功能，能够动态转换处理器核的供电电压，进一步降低了功耗；可以单独对 ADSP-BF535 Blackfin 处理器的每一个外设时钟进行控制，也进一步降低了功耗。

### 1. 全工作模式

在全工作模式中，PLL 提供最高的工作频率。这是正常工作状态，可以获得最高的性能。处理器核和所有工作的外设都以全速运行。

### 2. 主动工作模式

在主动工作模式中，PLL 处于工作状态，但是被旁路，采用输入时钟（CLKIN）产生处理器核的时钟（CCLK）和外设时钟（SCLK）。当 PLL 被旁路时，CCLK 的工作频率是 CLKIN 的 1/2。由于处理器核的工作频率是 CLKIN 的 1/2，可以极大地节约功耗。在该模式下，通过将 PLL 控制寄存器（PLL\_CTL）的 SSEL 域设置为特定的值，可以改变 PLL 的增值律。当处于主动模式时，支持对适当配置的 L1 存储器进行系统 DMA 访问。

### 3. 休眠工作模式

休眠工作模式将处理器核的时钟（CCLK）禁止，可以节约功耗。然而在该模式下，PLL 和系统时钟（SCLK）继续运行。任何中断，通常是某个外部事件或 RTC 行为，都将唤醒处理器。处于休眠模式时，任何中断的发出都将会使处理器识别出 PLL 控制寄存器（PLL\_CTL）中旁路位（BYPASS）的值。如果旁路被禁止，处理器将转换到全工作模式。处于休眠模式时，处理器不支持对 L1 存储器的系统 DMA 访问。

### 4. 深休眠工作模式

深休眠工作模式进一步降低了功耗，既禁止了 CCLK，也禁止了所有同步外设的 SCLK；异步外设仍然运行，例如 RTC，但是不能访问内部资源或外部存储器。只有发出复位中断（RESET），或者由 RTC 发出异步中断之后，才能退出这种节电模式。

当处于深休眠工作模式时，RESET 发出后，处理器将会获知 BYPASS 引脚的值。如果旁路被禁止，处理器将转换到全工作模式。如果旁路没有被禁止，处理器将转换到主动模式。当处于深休眠模式时，无论 BYPASS 引脚的值是多少，RTC 异步中断的发出都将使处理器转换到全工作模式。在这个模式下，会发送 DEEPSLEEP 输出。

## 15.2.18 工作模式和状态

处理器支持如下 3 种处理器模式：

- 用户模式；
- 管理模式；
- 仿真模式。

仿真和管理模式对内核资源的访问没有限制。用户模式对一些系统资源的访问有限制，因此可以提供一种受保护的软件环境。用户模式是应用程序的工作域，管理模式和仿真模式通常保留在操作系统的内核程序使用。

处理器模式由事件控制器来确定。当为一个中断进行服务时，例如不可屏蔽中断（NMI）或异常，处理器处于管理模式。当为一个仿真事件服务时，处理器处于仿真模式。如果不对任何事件提供服务，处理器处于用户模式。

## 习题

1. 列出 4 种类型的 DSP 消费类和通信类市场。
2. 在 C55x 中集成了几个 MAC 单元？

3. 描述 C55x 的 7 个流水线站。
4. 描述 C55x 中的 4 个主要功能模块。
5. 列出 C55x DU 中体系结构并行性的 5 个主要方面。
6. 列出 C55x 中 6 种先进的低功耗应用功能。
7. 列出 C55x 程序流单元的 5 个功能。
8. 在 Blackfin 处理器体系结构中有哪 3 个主要的功能?
9. 列出 4 个 Blackfin 系统外设。
10. 描述 Blackfin 的双 MAC 单元功能。
11. Blackfin 中 L1/L2 存储器系统的意义是什么?
12. 列出 Blackfin 处理器支持的 5 种类型的事件。
13. 在 Blackfin 中使用 DMA 控制的优点有哪些?
14. 描述 Blackfin 中 watchdog 定时器 (WDT) 的工作过程。
15. 列出 Blackfin 处理器的 4 种电源管理模式。

## 参考文献

- TMS320C55x™ DSP peripherals overview, reference guide.* Preliminary Draft. TI Literature Number: SPRU317G. February 2004.
- TMS320C55x™ DSP, programmer's guide, preliminary draft.* TI Literature Number: SPRU376A. August 2001.
- TMS320C55x™ DSP, functional overview.* TI Literature Number: SPRU312. June 2000.
- TMS320C55x™ DSP, CPU reference guide.* Literature Number: SPRU371F. February 2004.
- TMS320C55x™ technical overview.* TI Literature Number: SPRU393. February 2000.
- TMS320C55x™ DSP algebraic instruction set reference guide.* TI Literature Number: SPRU375G, October 2002.
- TMS320VC5507/5509 DSP analog-to-digital converter (ADC) reference guide.* TI Literature Number: SPRU586B. June 2004.
- TMS320VC55x01/5502 DSP direct memory access (DMA) controller reference guide.* TI Literature Number: SPRU613G. March 2005.
- TMS320VC55x10 DSP external memory interface (EMIF) reference guide.* TI Literature Number: SPRU590. August 2004.
- TMS320VC55x03/5507/5509 DSP external memory interface (EMIF) reference guide.* TI Literature Number: SPRU670A. June 2004.
- TMS320VC55x01/5502/5503/5507/5509 DSP inter-integrated circuit (I2C) module reference guide.* TI Literature Number: SPRU146D. October 2005.
- ADSP-BF535 Blackfin® embedded processor, data sheet.* REV. A. 2005, Analog Devices.
- ADSP-BF533 Blackfin® processor hardware reference.* Revision 3.2. July 2006, Part Number 82-002005-01, Analog Devices.
- Getting started with Blackfin® processors.* Revision 1.0. February 2005, Part Number 82-000850-01, Analog Devices.
- TMS320VC55x01/5502 DSP instruction cache reference guide.* TI Literature Number: SPRU630C. June 2004.
- TMS320VC55x01/5502 DSP timers reference guide.* TI Literature Number: SPRU618B. April 2004.
- TMS320VC55x01/5502 DSP universal asynchronous receiver/transmitter (UART) reference guide.* TI Literature Number: SPRU597B, November 2002. Revised March 2004.
- TMS320VC55x09 DSP multimediacard / SD card controller reference guide.* TI Literature Number: SPRU593. June 2003.
- TMS320VC55x07/5509 DSP universal serial bus (USB) module reference guide.* TI Literature Number: SPRU596A. June 2004.

# 嵌入式微控制器与处理器设计

## Embedded Microcontrollers and Processor Design

本书全面讲述了嵌入式微控制器的基础知识以及先进的设计方法。全书内容覆盖了RISC体系结构、数字信号处理、模糊逻辑和模数转换等主要概念，重点关注RISC相关设计。本书不是专门教授某个特定的微控制器设计方法，而是关注于从单片机到SoC系统芯片的RISC实现技术的整体变革，因此更适合用作高校电子电气工程、计算机以及工程技术类相关专业的教材，还可用作专业嵌入式微控制器设计人员的参考书。

### 本书特色

- 深入讲解微控制器、处理器设计以及相关技术。
- 通过RISC技术的工程开发，使读者深入理解设计中的主要理论。
- 内容新颖，包含了许多最新技术，如模糊逻辑以及MIPS、ARM和Tensilica的IP核设计。
- 讨论如何平衡处理器、存储器和软件三种技术，使学生深入理解半导体制造能力、分级存储结构和汇编器以及编译器优化等技术。
- 每章（除第1章外）后面都附有习题，用以复习本章的主要内容。



英文影印版  
ISBN: 978-7-111-29250-0  
定价: 49.00元



客服热线: (010) 88378991, 88361066  
购书热线: (010) 68326294, 88379649, 68995259  
投稿热线: (010) 88379604  
读者信箱: hzjsj@hzbook.com

PEARSON

[www.pearsonhighered.com](http://www.pearsonhighered.com)

华章网站 <http://www.hzbook.com>

网上购书: [www.china-pub.com](http://www.china-pub.com)

新华书店·金马

上架指导: 计算机 嵌入式系统

ISBN 978-7-111-32281-8



9 787111 322818

定价: 59.00元

[General Information]

书名=嵌入式微控制器与处理器设计

SS号=12767441